UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SISTEMA DE LECTURA DE CORREO ELECTRÓNICO PARA INVIDENTES

David Doyágüez Sánchez
Tutor: Germán Montoro Manrique

ABRIL 2019

SISTEMA DE LECTURA DE CORREO ELECTRÓNICO PARA INVIDENTES

AUTOR: David Doyágüez Sánchez
TUTOR: Germán Montoro Manrique

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Abril de 2019

Resumen (castellano)

Este Trabajo Fin de Grado consiste en un sistema de correo electrónico especialmente diseñado para usuarios invidentes. El trabajo se ha dividido en dos partes bien definidas:

Primero se ha diseñado e implementado una arquitectura que permite crear interfaces de usuario para invidentes. Se ha empaquetado esta arquitectura en una biblioteca de clases de forma que se pueda reutilizar el código. Esta biblioteca se ha diseñado de forma que sea multiplataforma. Se ha escogido Java como lenguaje principal y único de implementación debido a su alta disponibilidad en ordenadores personales y entornos tanto domésticos como profesionales.

Para validar esta arquitectura, se debía crear un programa lo suficientemente complejo como para poder demostrar que la biblioteca sirve a su propósito, por lo tanto se ha implementado un cliente de correo electrónico usando esta biblioteca, de forma que soporte la mayoría de las funcionalidades presentes en los clientes de correo electrónico actuales. Durante el desarrollo del cliente de correo electrónico, la biblioteca original fue modificada en varias ocasiones, de forma que fuera más potente y pudiera dar soporte a funcionalidades que no habían sido previstas en el diseño original.

Durante todo el desarrollo se han tenido en cuenta diversos estándares de accesibilidad, tanto en el diseño de la biblioteca como del cliente de correo electrónico, de forma que la interfaz resulte intuitiva.

Por último, se han ejecutado una serie de pruebas con usuarios que, aún no siendo invidentes, han permitido pulir aspectos importantes de usabilidad y corregir fallos que no se detectaron durante fases anteriores del desarrollo.

En definitiva, este trabajo presenta una arquitectura extensible y multiplataforma que permite crear interfaces de usuario para invidentes, validada mediante el desarrollo de un sistema de correo electrónico que cumple diversos estándares de accesibilidad.

Abstract (English)

This Bachelor Thesis consists in an e-mail client specifically designed for blind people. The project has been divided in two well differenced assets.

First of all, an architecture that makes possible creating user interfaces for the blind has been designed. This architecture was packaged in a class library so the code can be reused easily. This library is a multiplatform library, with Java being the main programming language used because of its large availability in personal computers and professional environments.

In order to validate this architecture and to show that the library serves to its purpose, a program enough complex should be created, so an email client with nearly all the capabilities of the current major email clients was implemented using the library. During the development of the email client, the original library was modified many times, so it became more powerful and support functionality that was not covered by the original design.

During the whole development various accessibility standards were considered, so theoretically the interface is intuitive.

Finally, some tests with users were performed, and altough the test users were not blind, those tests made possible to fix some major usability fails that could not be detected in previous development stages.

So, summing up, in this thesis, it is exposed an extensible, multiplatform architecture that allows the implementation of user interfaces for the blind, validated thanks to the development of an email client that meets a series of accessibility standards.

Palabras clave (castellano)

interfaz, invidente, arquitectura, correo, accesibilidad, usabilidad, biblioteca, java, multiplataforma,…

Keywords (inglés)

interface, blind, architecture, mail, accessibility, usability, library, java, multiplatform,…

Agradecimientos

Quisiera dar las gracias a la Universidad Autónoma en general por haberme enseñado tanto durante estos años. Ha sido duro pero ha merecido la pena.
Creo que no hay muchas personas que ante una idea loca como es este TFG, no sólo lo vieran viable, si no que además se sintieran entusiasmados por la misma; es por esto que me gustaría agradecer a Germán Montoro, tutor de este TFG, su apoyo y consejo durante la realización del mismo. Espero que sigamos trabajando juntos en el futuro. Todo esto no habría sido posible sin ti.
También me gustaría agradecer a Gonzalo Martínez, mi tutor de Grado, su apoyo durante toda la carrera.
Un abrazo también a Carlos Santacruz y Francisco Jurado, os recuerdo con cariño.
Creo que es justo mencionar a Parham Doustdar, programador iraní invidente cuyo blog me inspiró para la realización de este trabajo.
Recuerdo con especial cariño a todos, a todos mis amigos de la universidad, que han hecho de mi estancia aquí una época agradable y digna de recordar, dejando las *deadlines* imposibles y los exámenes insuperables a un lado.
Han sido unos compañeros de viaje imprescindibles Juan Alvear, Manuel Gómez y Jose Javier Martín. Lo hemos pasado bien y al final eso es lo importante.
También mencionar a mis amigos de fuera de la universidad.
Y por último pero no por ello menos importante (se trata de gente esencial), a mi familia:
Mi madre, mi padre y mi hermano me han aguantado con cariño en los peores momentos, así que el gracias más grande de todos es para vosotros.

INDICE DE CONTENIDOS

1 Introducción....1
1.1 Motivación 1
1.2 Objetivos 2
1.3 Organización de la memoria 2
2 Estado del arte....3
2.1 Sistemas lectores de pantalla 3
2.1.1 JAWS....3
2.1.2 NVDA....4
2.1.3 VoiceOver....4
2.1.4 Orca....4
2.2 Sistemas de texto a voz 4
2.2.1 Google Cloud Text To Speech API....4
2.2.2 Microsoft Azure Cognitive Services....5
2.2.3 Festival....5
2.2.4 Espeak....5
2.2.5 MaryTTS....5
2.3 Clientes de correo electrónico 5
2.3.1 Microsoft Outlook....6
2.3.2 Mozilla Thunderbird....6
2.3.3 Alpine....6
2.3.4 Gmail....7
2.4 Conclusiones 7
3 Diseño....9
3.1 Introducción 9
3.2 Diseño de la arquitectura 9
3.2.1 Motivación....9
3.2.2 Diseño general de la arquitectura....10
3.2.3 Diseño del subsistema de texto a voz....10
3.2.4 Diseño del subsistema de componentes....10
3.2.5 Diseño del subsistema principal....12
3.3 Diseño del cliente de correo electrónico 13
3.3.1 Subsistema de lectura de emails....13
3.3.2 Subsistema de escritura de emails....14
3.3.3 Subsistema de contactos....14
3.3.4 Subsistema criptográfico....14
3.3.5 Subsistema de autentificación....14
3.3.6 Subsistema principal....14
4 Desarrollo....15
4.1 Introducción 15
4.2 Desarrollo de la arquitectura 15
4.2.1 Desarrollo general de la arquitectura....16
4.2.2 Desarrollo del sistema texto a voz....17
4.2.3 Desarrollo del sistema de componentes....18
4.2.4 Desarrollo del sistema principal....20
4.3 Desarrollo del cliente de correo electrónico 22
4.3.1 Módulo de autentificación....23
4.3.2 Módulo criptográfico....24

4.3.3 Módulo de contactos....24
4.3.4 Módulo de lectura de emails....25
4.3.5 Módulo de escritura de emails....26
4.3.6 Módulo principal....26
4.3.7 Diagrama de navegación....27
5 Integración, pruebas y resultados....29
5.1 Integración con el patrón modelo-vista-controlador 29
5.2 El proceso de desarrollo incremental iterativo 29
5.2.1 Iteración 1. Módulos de autentificación, principal y de lectura....29
5.2.2 Iteración 2. Módulos de contactos, encriptación y escritura....30
5.2.3 Iteración 3. Mejoras en el módulo principal y de encriptación e interconexión de los módulos de lectura y escritura....31
5.3 Pruebas con usuarios 31
5.3.1 Usuario 1....32
5.3.2 Usuario 2....34
5.3.3 Comparación de resultados del usuario 1 frente al usuario 2....35
6 Conclusiones y trabajo futuro....37
6.1 Conclusiones 37
6.2 Trabajo futuro 38
Referencias....39
Glosario....41
Anexos....I
6.3 Artículo enviado a Interacción 2019 I

INDICE DE FIGURAS

Figura 1: Diagrama de la arquitectura Blue....16
Figura 2: Diseño del sistema TTS en forma de diagrama de clases parcial....18
Figura 3: Diseño del sistema de componentes en forma de diagrama de clases parcial....20
Figura 4: Diseño del sistema principal en forma de diagrama de clases parcial....21
Figura 5: Diseño del cliente de correo electrónico en forma de diagrama de clases....22
Figura 6: Diagrama de navegación del cliente de correo RedMail....27

INDICE DE TABLAS

Tabla 1: Desempeño en pruebas del usuario 1....32
Tabla 2: Desempeño en pruebas del usuario 2....34
Tabla 3: Desempeño del usuario 2 frente al usuario 1....35

1 Introducción

1.1 Motivación

Durante las últimas décadas estamos viviendo una serie de cambios tecnológicos y sociales equiparables a la Revolución Industrial. Nos estamos adentrando como especie en la Era de la Información. Estos cambios, con sus luces y sus sombras, están modificando la manera en la que las personas interactuamos con el mundo. Se han tendido cables que nos permiten comunicarnos al instante con continentes al otro lado del mundo, podemos realizar cálculos complejos en segundos, enterarnos de sucesos lejanos de inmediato, hemos ganado velocidad, el ser humano se ha vuelto un ser rápido.

Avanzamos a tal velocidad, que en ocasiones no nos paramos a pensar porqué avanzamos. La ingeniería tiene como propósito resolver los problemas y poder satisfacer las necesidades de la gente [1], y en efecto es lo que hace. El problema reside en que este progreso, al ser tan veloz, deja atrás a muchas personas que pueden beneficiarse enormemente de los nuevos avances tecnológicos. En parte tiene sentido, porque para que haya velocidad y los nuevos avances lleguen al mayor número de personas en el menor tiempo posible, todo se diseña, efectivamente, para que resulte cómodo y usable para la mayoría. Pero ¿qué sucede con las minorías?. Quedan relegadas a un segundo plano.

En concreto, me refiero a aplicaciones y programas que no se diseñan teniendo en cuenta estándares de accesibilidad. Al no realizar este diseño o implementación teniendo en cuenta que existen personas que no funcionan igual que la mayoría (personas con diversidad funcional), esta minoría queda excluida injustamente del progreso, cuando precisamente ellos son los que mayor ventaja podrían obtener del mismo (y obtienen de forma efectiva cuando alguien se para un momento a pensar en ellos y a desarrollar avances específicos) [2]. Aún así, el problema de esto es que aunque alguien empiece a desarrollar tecnología específicamente accesible, en un mundo en el que prima la velocidad el tiempo es el elemento más valioso, por lo tanto el tiempo que se tarda en crear esta tecnología debe ser remunerado de alguna forma, haciendo en muchos casos que el precio de la tecnología accesible existente sea prohibitivo. Uno de los problemas que vamos a abordar en este proyecto es ese, que tecnología que se espera que sea accesible se vuelva en muchos casos inaccesible precisamente por una cuestión económica. Por esto, un proyecto académico como este es una gran manera de emplear el tiempo, para poner la ingeniería informática al servicio de una minoría que puede aprovecharse enormemente del progreso, como son las personas invidentes.

En concreto se ha desarrollado una biblioteca de clases extendible, sólida y multiplataforma que permita una implementación rápida de interfaces de usuario para invidentes. Se ha escogido Java como lenguaje principal.

Para validar esta biblioteca se ha construido con ella un sistema de correo electrónico que implementa la mayoría de las funcionalidades presentes en los clientes de correo electrónico actuales.

1.2 Objetivos

Este proyecto tiene dos objetivos bien definidos:

1. Conseguir una arquitectura sólida, estable, extendible, multiplataforma y bien documentada que permita la creación fácil y rápida de interfaces de usuario para invidentes que cumplan diversos estándares de accesibilidad.
2. Validar esta arquitectura mediante la implementación de un programa complejo como es un cliente de correo electrónico específicamente diseñado para invidentes y que cumple con diversos estándares de accesibilidad.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte**
- **Diseño**
- **Desarrollo**
- **Integración, pruebas y resultados**
- **Conclusiones y trabajo futuro**

2 Estado del arte

2.1 Sistemas lectores de pantalla

Un lector de pantalla es un programa que permite a un usuario invidente utilizar una interfaz gráfica mediante una lectura de la pantalla. Gracias a un sintetizador de voz, el lector de pantalla lee lo que se muestra por pantalla proporcionando una descripción de los elementos de cada programa y permitiendo al usuario conocer la interfaz, en vez de por medios visuales, por medios auditivos.

Los lectores de pantalla son una gran herramienta imprescindible para los usuarios invidentes, pero el problema es que la mayoría no son portables ni multiplataforma, y no dejan de ser un "parche" a un problema más profundo, que es que el diseño de los programas no está adaptado a las personas invidentes.

Si es cierto que los programas pueden estar diseñados para funcionar con un lector de pantalla (en ocasiones no es así, por ejemplo las interfaces gráficas Java requieren de un componente especial instalado en el ordenador del usuario denominado Java Accessibility Bridge), pero la experiencia que ofrecen es subóptima, por ejemplo en programas cuyas interfaces son muy complejas es común que los usuarios pierdan el foco de la aplicación o incluso no puedan usarla, como es el caso de Spotify [2]. Además, aunque se intente adaptar la aplicación a personas invidentes con las tecnologías de accesibilidad actuales, la experiencia del usuario nunca será tan buena como con una interfaz creada específicamente para una persona invidente.

El foco es un concepto importante en todo esto. Tiene el foco el elemento de la interfaz que está captando la atención del usuario en ese momento, por ejemplo, en una interfaz gráfica cuando un usuario está escribiendo en un cuadro de texto, es el cuadro de texto el que posee el foco.

Los lectores de pantalla actúan describiendo de forma genérica el elemento que posee el foco, y cuando este foco cambia (por medio de la tecla de cambio de foco, que suele ser el tabulador o atajos aceleradores si la aplicación los tiene implementados) se lee el nuevo elemento enfocado. En la mayoría de los toolkits modernos para implementar interfaces gráficas de usuario, podemos encontrar formas de establecer un orden de foco, es decir, el orden en el que los elementos serán presentados al usuario, pero en ocasiones es complicado encontrar un orden de enfocado adecuado para la tarea en caso de que la interfaz sea compleja (porque la interfaz no está diseñada para ser secuencial), y eso en el mejor de los casos. En el peor de los casos el desarrollador ni siquiera pensará en hacer accesible la aplicación. A continuación, vamos a hacer una breve comparativa de los principales lectores de pantalla en PC, y podremos comprobar que presentan problemas de portabilidad o precio.

2.1.1 JAWS

JAWS, *Job Access With Speech* es el lector de pantalla más usado por invidentes según un estudio de WebAIM, alcanzando una cuota de mercado del 46.6% en 2017 [3].

Creado por la empresa Freedom Scientific, este lector de pantalla proporciona salida hablada mediante un sistema de texto a voz (TTS) o salida Braille. La licencia más barata es de 90 dólares al año, siendo posible comprar una licencia de por vida por 1000 dólares. Funciona en Windows exclusivamente, por lo que tiene un problema de portabilidad.

2.1.2 NVDA

NVDA, *NonVisual Desktop Access* es la alternativa libre a JAWS, es gratuito y tuvo una cuota de mercado del 31.9% en 2017 [3], creado por NV Access y mantenido por múltiples personas de diferentes nacionalidades, incluyendo también patrocinio de Google, Adobe y Microsoft entre otras [4]. Proporciona salida mediante texto a voz o salida Braille. Funciona en Windows exclusivamente, por lo tanto presenta problemas de portabilidad.

2.1.3 VoiceOver

VoiceOver es el lector de pantalla incorporado en iOS y macOS, viene preinstalado en los dispositivos de Apple y ha sido desarrollado por la misma, por lo que está altamente integrado en el sistema. Tuvo una cuota de mercado del 11.7% en 2017 [3]. Proporciona salida mediante texto a voz o salida Braille [5].
Funciona en dispositivos Apple exclusivamente, por lo que presenta problemas de portabilidad.

2.1.4 Orca

Orca es el lector de pantalla incorporado por defecto en sistemas GNU/Linux. Al ser parte del proyecto GNOME, es software libre y gratuito. Se desconoce la cuota de mercado de Orca, pero al ser el único lector de pantalla disponible en GNU/Linux, podemos estimar que habrá sido como máximo de un 1.4% aproximadamente en 2017 [3]. Proporciona salida mediante texto a voz o salida Braille [6].
Funciona en sistemas operativos tipo Unix, como BSD o Linux, así que también presenta problemas de portabilidad.

2.2 Sistemas de texto a voz

Un sistema de texto a voz, *text to speech* (TTS), también conocido como sintetizador de voz, es un software que permite dado un texto, emitir un audio en el que suena ese texto dictado por una voz sintética. Es un elemento fundamental en los lectores de pantalla, pues es el componente que permite hacer efectiva la emisión de los datos al usuario mediante el canal auditivo.
Actualmente existe una gran variedad de sistemas de texto a voz, pero, como veremos a continuación, no es tan sencillo encontrar soluciones multiplataforma que puedan usarse sin coste. Con el auge de las redes neuronales, la computación en la nube y el Big Data, grandes empresas como Google, Microsoft y Amazon ofrecen servicios de texto a voz en la nube, con una calidad y naturalidad que demuestra ser muy superior a los sistemas de texto a voz offline tradicionales.
A continuación vamos a realizar una breve comparativa entre los principales sistemas de texto a voz que podemos encontrar actualmente.

2.2.1 Google Cloud Text To Speech API

Es el sistema de texto a voz en la nube de Google. Requiere de conexión a internet constante para enviar peticiones a la API REST de Google y recibir el archivo con la grabación de la voz. Es gratuito hasta que se alcanzan los cuatro millones de caracteres enviados, cuando se alcanzan son cuatro dólares por millón de caracteres [7]. Se trata de

software no libre, el desarrollador debe tener una cuenta de Google Cloud para poder obtener una API key que permita usar el servicio, que está sujeto a estas limitaciones de escalabilidad, es decir, aunque al principio con una base de usuarios pequeña no suponga un coste, más adelante podría ser un problema a la hora de mantener un software que use este servicio.

2.2.2 Microsoft Azure Cognitive Services

Es el sistema de texto a voz en la nube de Microsoft. Requiere de conexión a internet constante para enviar peticiones a su API REST y recibir el archivo de voz. Es gratuito hasta que alcanzas un límite de cinco mil transacciones (llamadas a API) al mes, a partir de entonces son veinte dólares por minuto hablado. Funciona también por medio de una API key [8].

2.2.3 Festival

Es un sistema de texto a voz libre (licencia X11) y gratuito que permite su uso en aplicaciones tanto personales como comerciales. Ofrece una plataforma para construir sistemas de sintetización de voz, aparte de poder ser usado a través de APIs para distintos lenguajes:

1. Desde la línea de comandos.
2. A través de un intérprete de comandos Scheme.
3. Desde una biblioteca C++.
4. A través de Java.

Soporta oficialmente sistemas tipo Unix, aunque se han creado binarios que pueden ser usados en Windows de forma no oficial y no tan estable como en Unix [9].

2.2.4 Espeak

Es el sistema de texto a voz usado por el lector de pantalla NVDA. Es software libre (licencia GPL) que permite uso no comercial. Funciona en Windows y Linux, se puede interactuar a través de SAPI5 (Speech API) en Windows, y existe una biblioteca para Linux [10].

2.2.5 MaryTTS

Es un sistema de texto a voz con licencia LGPL desarrollado por el DFKI (Centro Alemán de Investigación de Inteligencia Artificial) e implementado completamente en Java, es un sistema multiplataforma diseñado para que sea relativamente sencillo agregar nuevas voces e idiomas al sistema. Este sintetizador soporta dos tipos de voces, HMM (Hidden Markov Model) y Unit Selection [11]. Es el sintetizador que se ha escogido para este proyecto por la naturalidad de sus voces por defecto y su capacidad multiplataforma. También se ha considerado el hecho de que es un sistema bien mantenido y en constante desarrollo e innovación (lleva presente desde el año 2000).

2.3 Clientes de correo electrónico

En la Era de la Información en la que nos encontramos, la comunicación por internet se ha vuelto casi esencial en la sociedad actual. Según un estudio de Eurostat, el 72% de

personas entre los 16 y los 78 años usaban internet para enviar o recibir emails en 2017 [12]. El correo electrónico, lejos de desaparecer, es una herramienta usada tanto en entornos domésticos como profesionales, muchas redes sociales y aplicaciones hacen un uso intensivo del correo electrónico como medio de registro, por ejemplo para enviar links de activaciones de cuenta, recuperar contraseñas, o también para mandar breves resúmenes de la actividad cuando has estado un tiempo sin usarlas.
El correo electrónico es una tecnología de tipo cliente servidor, es decir, el usuario ejecuta un programa cliente que envía instrucciones al servidor para realizar lectura y envío de correo. El servidor suele ser propiedad de la empresa con la que tengamos el correo electrónico.
Por tanto, el programa que debe ser accesible es el cliente de correo electrónico. A continuación vamos a realizar una breve comparativa de los clientes de correo actuales para PC en clave de accesibilidad y funcionalidades básicas.

2.3.1 Microsoft Outlook

Es el cliente de correo electrónico no web más usado, tiene soporte para IMAP, un sistema de contactos, soporte para SSL, visión de conversación, búsqueda y operaciones básicas como reenviar, poner en copia, copia oculta…
En su versión Express (ahora obsoleta) tenía un gran soporte para lectores de pantalla [13], pero con la llegada de nuevas versiones, el programa se ha ido haciendo cada vez menos amigable para los lectores de pantalla y por lo tanto menos accesible.

2.3.2 Mozilla Thunderbird

Thunderbird es el cliente de correo de Mozilla, tiene unas capacidades similares a Outlook, y tenía un buen soporte para ser usado con lectores de pantalla en versiones antiguas [13], pero de nuevo, con la llegada de nuevas versiones se ha ido haciendo cada vez más complicado de utilizar y menos soportable por lectores de pantalla. De todos los clientes de correo que analizamos en este trabajo, consideramos Thunderbird el más accesible y por lo tanto el programa “rival” al que superar en usabilidad (cosa que, como veremos más adelante, se vuelve factible gracias a la arquitectura que se ha desarrollado en este trabajo). Aún siendo el más accesible, la complejidad de su interfaz impide que el uso de este programa sea óptimo. Contempla una gran cantidad de opciones y funcionalidades, pero es un cliente de correo con una interfaz compleja precisamente por eso, y por lo tanto, es subóptimo en términos de accesibilidad.

2.3.3 Alpine

Alpine es un cliente de correo de terminal que permite navegar por su interfaz mediante atajos de teclado, es posible usarlo totalmente con el teclado, lo que lo hace un programa muy accesible [13]. También permite configurarlo de modo que se usen otros programas para editar texto o mostrar adjuntos (característica que, si se piensa bien, es genial porque permite por ejemplo ejecutar un programa externo que describa el texto de una imagen o abra ficheros PDF con el programa accesible favorito del usuario).
Cabe destacar que Alpine hace un uso intensivo de los menús, y en cierta manera nos hemos inspirado en esto a la hora de realizar el diseño de la arquitectura como veremos más adelante.
También, al ser un programa de terminal tiene un problema con algunos lectores de pantalla, en el sentido de que por la forma en la que interactúan los lectores de pantalla con el terminal no es posible detener la locución del lector de pantalla a demanda, y veremos más adelante qué implicaciones ha tenido eso en el desarrollo de este trabajo y porqué es

un dato tan importante. En cualquier caso, Alpine es un cliente de correo subóptimo en términos de accesibilidad por los bugs que presenta al interactuar con lectores de pantalla.

2.3.4 Gmail

Con más de un billón de usuarios según TechRadar, es el cliente web más usado [14].
Gmail cuenta con una aplicación para Android que no vamos a tener en cuenta en este trabajo debido a que nos centramos en clientes de correo que se usan desde PC. La interfaz web de Gmail, al ser una interfaz web tiene cierto soporte para lectores de pantalla, pero el posicionamiento de los elementos en esta interfaz resulta incómodo para los usuarios invidentes, precisamente porque se presenta mucha información al usuario en un mismo menú. El hecho de que haga un uso intensivo de AJAX puede resultar un impedimento también para ciertos lectores de pantalla, y puede ocasionar pérdidas de foco y estrés en el usuario.

2.4 Conclusiones

Como hemos podido deducir de nuestra investigación, los lectores de pantalla que hemos visto antes presentan un problema generalizado, la falta de portabilidad. Esto hace que los usuarios se vean restringidos a un único sistema operativo, que suele ser Windows (los dos principales lectores de pantalla, JAWS y NVDA funcionan exclusivamente bajo Windows). Para agravar el problema, las versiones que mejor tienden a funcionar suelen ser versiones antiguas de programas (como por ejemplo el caso de Outlook), lo que hace que muchos usuarios invidentes usen versiones antiguas de programas o, en casos extremos, versiones antiguas de sistemas operativos como Windows XP; esto representa un gran problema de seguridad.
Gracias a la arquitectura que se presenta en este trabajo, podemos no sólo atajar el problema de la portabilidad (aumentando también la seguridad de los usuarios), si no que podemos mejorar ampliamente el sistema tradicional de lectores de pantalla eliminando el intermediario, es decir, eliminando el lector de pantalla. Al ser la propia aplicación la que da la información ya adaptada al canal auditivo, también solucionamos los problemas de usabilidad que tienen los usuarios, en concreto las pérdidas de foco (desorientación).
Esto se consigue contextualizando mucho la información dada al usuario, cosa que no es posible que haga un lector de pantallas porque es una solución genérica, frente a la solución propuesta que es una arquitectura que permite generar soluciones específicas adaptadas al programa que se va a crear.

Con respecto a los clientes de correo, en la misma línea presentan problemas de portabilidad, siendo Thunderbird el más accesible de todos. El problema que presenta es que la complejidad de su interfaz hace que el desempeño de los usuarios sea subóptimo. Por supuesto, el usuario acaba acostumbrándose, pero se le podría dar una experiencia de usuario mucho más amigable mediante un sistema adaptado que el usuario pudiera manipular y de esta manera conseguir un desempeño y una satisfacción mucho mayores que con un cliente de correo convencional y un lector de pantallas.

3 Diseño

3.1 Introducción

Este trabajo ha sido dividido en dos partes esenciales: se ha desarrollado una arquitectura que permite programar interfaces de usuario especialmente adaptadas para invidentes, y se ha desarrollado a modo de validación de la arquitectura un cliente de correo electrónico con la biblioteca de clases obtenida de implementar dicha arquitectura.
Por tanto, para la obtención de estos dos elementos, ha habido que hacer un proceso de diseño independiente para ambos.

3.2 Diseño de la arquitectura

3.2.1 Motivación

El objetivo principal de la arquitectura es el de permitir programar de forma sencilla interfaces de usuario especialmente diseñadas para su uso por personas invidentes.
Para comprender esto, tenemos que entender las interfaces de usuario actuales; en general, la mayoría de interfaces de usuario son interfaces gráficas de usuario (GUIs).
Estas interfaces gráficas son usadas mediante el sentido de la vista, recibiendo input por parte del usuario a través del teclado, un ratón, o una pulsación táctil dependiendo del dispositivo que se esté utilizando, y recibiendo output el usuario a través de la pantalla.
Como el colectivo de usuarios para los que estamos desarrollando esta arquitectura carecen de visión, es necesario hacer un cambio en el paradigma de las interfaces de usuario.
Para una persona vidente, resulta sencillo acceder a toda una serie de componentes, comúnmente denominados *widgets* mediante el ratón. Este acceso es un acceso visual, por ejemplo el usuario vidente ve un botón, que tiene una forma, con su función definida probablemente mediante texto que el usuario ve, y es gracias a esa visión que puede tomar la decisión de mover el cursor por la pantalla (viendo en cada momento dónde está el cursor) y hacer click en el botón.
Un usuario invidente no es capaz de hacer esto. Lo que hace un usuario invidente con un lector de pantallas estándar es que, al no poder usar el sentido de la vista, usa el sentido del oído. Esto es, el lector de pantallas mediante un sistema de texto a voz lee cada *widget*.
Para eso, existe el concepto de foco. Tiene el foco el elemento que va a ser usado por el usuario en un momento dado. Cuando un elemento toma el foco (normalmente en los lectores de pantalla más populares se cambia el foco de un elemento al siguiente presionando el tabulador, y al anterior presionando mayúsculas y tabulador a la vez), el lector de pantalla identifica el *widget* que ha tomado el foco y da una descripción del elemento. Teniendo ya una descripción del elemento, el usuario puede tomar una decisión de usarlo o no. Al usarlo desencadenará una acción que hará que el lector de pantallas informe al usuario del cambio etc.
Lo que permite esta arquitectura es precisamente eso, pero de forma nativa a la aplicación que se esté desarrollando con esta arquitectura, es decir, eliminamos el intermediario (el lector de pantallas) de forma que la propia aplicación creada con la arquitectura sea capaz por si misma de proporcionar esta interacción no visual sin la necesidad de depender de un lector de pantallas. Esto permitirá a la aplicación estar mucho más contextualizada (las descripciones no tendrán porqué ser genéricas como en el caso de los lectores de pantalla) y mejor organizada para su uso por una persona invidente, porque la propia arquitectura obliga a establecer un orden en los componentes, y como veremos más adelante, cumple diversos criterios de accesibilidad que una aplicación corriente no tiene porqué.

La arquitectura, que se ha denominado Blue (BLind User InterfacE), ha sido empaquetada en una biblioteca denominada libblue.

3.2.2 Diseño general de la arquitectura

Teniendo ya claro que el objetivo de la arquitectura es dar una funcionalidad superior a la de un lector de pantalla para cada aplicación que se desarrolle mediante la misma, cabe destacar que aparte de la portabilidad y la flexibilidad que dará al usuario una aplicación creada con la arquitectura, también permitirá crear aplicaciones que estén específicamente adaptadas a la tarea que el usuario deba realizar con ellas, esto es, al estar intrínseca la interfaz auditiva en la aplicación y prescindir del elemento "conversor" (un lector de pantalla, al final lo que hace es convertir información visual ya existente a información auditiva) se permite una mayor flexibilidad en la interacción con la aplicación que la que puede producir un lector de pantalla. Esto es lógico, porque el lector de pantalla está diseñado para funcionar con el máximo número de aplicaciones creadas para videntes posible, y la arquitectura está diseñada para que la propia aplicación se adapte al usuario invidente, es decir, la información que genera una aplicación creada con la arquitectura ya está adaptada al canal que está usando el usuario, mientras que el lector de pantalla genera esa adaptación algorítmicamente.
Para proporcionar toda esta funcionalidad, se han previsto varios subsistemas:

- Subsistema de texto a voz
- Subsistema de componentes
- Subsistema principal

3.2.3 Diseño del subsistema de texto a voz

El subsistema de texto a voz es el encargado de dar voz a la aplicación. Todo lo que suceda en la aplicación será descrito por el sistema de texto a voz. Es el subsistema que va a realizar la interacción efectiva con el usuario. Está diseñado de forma que sea sencillo añadir soporte para nuevos sistemas de texto a voz.
La única condición que debe darse es que el sistema de texto a voz soporte que se interrumpa una locución activa.
Esto es necesario para permitir una rápida navegación por la interfaz. Dada la naturaleza secuencial de este tipo de interfaces (recordemos que el foco se toma de forma secuencial mediante pulsaciones del tabulador), es necesario que el usuario identifique rápidamente los componentes. Por lo tanto, si a mitad de la lectura de un componente el usuario decide enfocar el siguiente elemento, no debe tener que esperar a que el programa lea el componente, si no que el siguiente componente que acaba de tomar el foco debe empezar a ser leído de inmediato, de ahí la necesidad de que el sistema de texto a voz soporte que se le interrumpa.

3.2.4 Diseño del subsistema de componentes

En esta arquitectura, para poder implementar aplicaciones que tengan una funcionalidad y que hagan cosas, necesitamos que efectivamente exista una interacción por parte del usuario, es decir, un input del usuario. Las acciones que llevaría a cabo un usuario vidente en una interfaz gráfica las podría realizar gracias a que existen ventanas, botones, cuadros de texto… Por lo tanto es necesario tener un reemplazo adecuado de estos elementos, porque al fin y al cabo son a los que están acostumbrados los usuarios invidentes de lectores de pantalla y por lo tanto a lo que mejor se van a adaptar. En nuestra arquitectura, llamamos a estos elementos componentes.

Al no tener canal visual para presentar la información de la interfaz, es necesario encontrar otra manera de organizar los componentes.
De esta manera, en vez de tener ventanas que son esencialmente cuadradas, tendremos listas de componentes, que hemos denominado menús (recordemos los menús del cliente de correo electrónico Alpine y el buen resultado que daban en términos de accesibilidad).
Tenemos también botones análogos a los botones de una interfaz gráfica. Cuando un botón tiene el foco, al pulsar Enter (o otra tecla que puede definir el programador que use la arquitectura) realiza una acción definida mediante una *callback*.
Los botones pueden conformar listas de botones. Las listas de la arquitectura en general son una forma de añadir dimensionalidad a la interfaz. Por ejemplo, podríamos tener una lista de carpetas y al lado una lista de archivos de esa carpeta, y poder recorrer rápidamente las carpetas mediante las flechas del teclado (*arrow keys*).
Un componente importante que se ha introducido es el *input buffer*. Un *input buffer* permite al usuario escribir un texto. Para conseguir que el usuario pueda usar atajos de teclado sin interferir en la escritura ni tener que desplazarse a otro elemento para realizar el atajo, se ha definido un sistema de escritura modal, esto es, cuando el búfer obtenga el foco, por defecto estará en modo descripción (se leerá al usuario la función del búfer y qué información tiene introducida). Si el usuario pulsa la tecla Control, entonces el búfer pasará a estar en modo edición (el usuario es informado de todo esto), y el usuario puede teclear la información que desee introducir. En modo descripción se ha habilitado la posibilidad de copiar y pegar información del búfer y hacia el búfer. Pulsando la tecla C se copia información del búfer al portapapeles, y pulsando la tecla P se pega información del portapapeles en el búfer.
Se ha introducido también un componente etiqueta, que es similar a las etiquetas de una interfaz gráfica, es decir, al igual que en una interfaz gráfica se muestra un texto, con nuestra arquitectura cuando una etiqueta toma el foco el sistema de texto a voz enuncia un texto. Gracias a este componente podemos notar la diferencia conceptual entre una interfaz gráfica y una interfaz auditiva como la nuestra; esto es, cuando una etiqueta gráfica cambia el usuario puede darse cuenta de inmediato. Esta afirmación en realidad no es del todo correcta, porque en una interfaz gráfica el usuario se da cuenta cuando posa su vista sobre la etiqueta. Podemos decir entonces que es de nuevo una cuestión de acceso. El usuario vidente se da cuenta de que la etiqueta ha cambiado porque ha accedido a la información, y el usuario invidente que use nuestra arquitectura se dará cuenta de que la etiqueta ha cambiado en el momento en el que el TTS lea el contenido de la etiqueta, es decir, en el momento en el que se enfoque la etiqueta. Esto es importante porque nos permite establecer un sistema de *callbacks* por el que una etiqueta tome valor en el momento en el que es enfocada. De este modo, podemos mantener información dinámica accesible de forma sencilla (es más sencillo definir una *callback* que cambiar desde varias partes del código de la aplicación el valor de la etiqueta).
De igual manera que los botones formaban listas de botones, las etiquetas conforman listas de etiquetas. Esto es muy útil para leer textos grandes. Pongámonos en el supuesto de que se va a leer un texto muy largo, como por ejemplo, un correo electrónico largo. Con una etiqueta, el usuario no puede navegar por las frases porque el TTS se limita a leer el texto, pero si creamos una lista de frases, permitimos al usuario navegar por ellas mediante las flechas del teclado, al igual que se navega por una lista de botones.
Sin embargo, pensemos en el ejemplo del correo largo. Tal y cómo habíamos definido las listas, se leería la primera frase y el usuario podría pulsar las flechas del teclado para oir la siguiente frase. Esto no aporta una continuidad, es decir lo óptimo es que cuando se empiece a escuchar un texto, se escuche entero, y que la opción de navegar por las frases

sea, en efecto, una opción. Para conseguir este comportamiento se ha implementado una lista de etiquetas especial que detallaremos más adelante.
Por último, se ha añadido un componente que imita a las cajas de confirmación (*checkbox*) de las interfaces gráficas. Este componente informa de su estado (activado o desactivado) de forma contextual a la funcionalidad que activa o desactiva, dando la opción al usuario de cambiar ese estado al otro estado posible de forma amigable, clara y contextualizada, y además informa al usuario del estado al que ha cambiado para hacer que la toma de decisiones sea más rápida.

3.2.5 Diseño del subsistema principal

El sistema principal ha sido diseñado de forma que sea altamente modular. Como estamos proporcionando una biblioteca que se usará para crear aplicaciones muy variadas, podríamos tener situaciones en las que los componentes implementados por defecto en la arquitectura no sean suficientes como para poder crear toda la funcionalidad necesaria para el programa que se vaya a implementar.
Es por esto que el diseño permite crear nuevos componentes que sean integrables con la arquitectura sin tener que modificar la misma.
Lo mismo sucede con los sistemas de texto a voz, cada vez tenemos sistemas de texto a voz más potentes y agradables al oído, y como queremos que esta arquitectura sea válida durante mucho tiempo, hemos proporcionado un mecanismo para poder acoplar fácilmente nuevos sistemas de texto a voz sin necesidad de modificar la arquitectura.
Se ha proporcionado la posibilidad de cambiar el orden de los componentes en tiempo de ejecución. Esto responde a la dificultad de crear una aplicación que sea agradable al uso para múltiples personas. No es complicado imaginar un escenario en el que el flujo de la aplicación (es decir, el orden en el que el usuario enfoca los componentes) no resulte el adecuado para alguna persona en particular. Con nuestra arquitectura, damos la capacidad al usuario de poder cambiar ese orden en tiempo de ejecución, de forma que sea el mismo usuario el que decida cómo es su interfaz. Al fin y al cabo, nadie mejor que el usuario conoce sus necesidades, y lo que conseguimos con esto es proporcionar al usuario esa capacidad de decisión. También permitimos guardar la configuración que ha realizado el usuario sobre la interfaz de forma sencilla (mediante una combinación de teclas, Alt + Enter), y que la próxima vez que el usuario ejecute la aplicación, esa configuración se cargue de forma transparente.
En definitiva, con esta arquitectura apostamos por darle la máxima autonomía al usuario con el mínimo esfuerzo por parte del programador que la use de base para sus aplicaciones.

3.3 Diseño del cliente de correo electrónico

El cliente de correo electrónico ha sido diseñado con dos objetivos principales:

1. Proporcionar un cliente de correo electrónico con la mayoría de la funcionalidad que incorporan los clientes de correo electrónicos modernos y que pueda usarse en cualquier ordenador de sobremesa desde un dispositivo de almacenamiento USB, de forma que el usuario pueda llevarlo consigo, y que cumpla ciertos criterios mínimos de seguridad esperables en un programa moderno.
2. Validar la arquitectura creando un programa con una interfaz compleja como es un cliente de correo electrónico, de forma que quede demostrado que la arquitectura es lo suficientemente potente como para soportar la creación de interfaces de usuario complejas.

El cliente de correo ha sido dividido en varios módulos:

- Subsistema de lectura de emails
- Subsistema de escritura de emails
- Subsistema de contactos
- Subsistema criptográfico
- Subsistema de autentificación
- Subsistema principal

Cada subsistema ha sido diseñado para que pueda ser modificado sin tener que realizar grandes cambios en los demás. A continuación pasamos a describir la funcionalidad de cada uno:

3.3.1 Subsistema de lectura de emails

Permite al usuario leer emails, identificar el remitente, identificar todos los destinatarios, soporta lo que se conoce como visión de conversación (y su propio nombre anuncia un problema), esto es, si un correo recibido es respuesta a otro correo, este subsistema permite identificarlo y navegar hacia el correo que mandó el usuario hasta llegar al primer correo que originó las consecuentes respuestas. Esta navegación permite retorno, es decir, puedes navegar por el hilo de la conversación hacia arriba (hasta llegar al original) o hacia abajo (hasta volver al primer correo que se leyó).

Soporta navegar por el cuerpo del email rápidamente mediante las flechas del teclado, de forma que al pulsar esas teclas deje de leer la frase actual y comience a leer de inmediato la frase siguiente.

Permite identificar hipervínculos y conseguir que el usuario pueda copiarlos fácilmente al portapapeles para su posterior visionado en un navegador (en caso de que disponga de un lector de pantalla externo) o cualquier otro uso.

Permite identificar archivos adjuntos y descargarlos a la carpeta que contiene el programa; esto se ha hecho así para evitar al usuario tener que borrar el archivo adjunto una vez lo haya leído, dada la naturaleza portable del cliente de correo es probable que se use en equipos que no sean propiedad del usuario.

Permite acceder al subsistema de escritura de emails mediante los botones de reenvío y respuesta, evitando al usuario tener que escribir el cuerpo o la dirección del destinatario de nuevo respectivamente.

Este subsistema interactúa con el subsistema de contactos permitiendo sustituir una dirección de correo almacenada en contactos por el nombre del contacto que se haya dado a ese correo. También se integra parcialmente con el sistema de contactos de Gmail.

3.3.2 Subsistema de escritura de emails

Permite al usuario escribir emails, poner correos en copia y en copia oculta, y agregar adjuntos.
Este subsistema proporciona un autocompletado a la hora de escribir la dirección de los destinatarios, de forma que no es necesario recordar la dirección exacta.
Permite pegar texto en el cuerpo del mensaje (por ejemplo para pegar hipervínculos) y agregar archivos, siendo posible navegar por el sistema de archivos del equipo desde el propio cliente de correo.

3.3.3 Subsistema de contactos

Permite al usuario agregar contactos, esto es, dar un nombre corto y fácilmente recordable a una dirección de correo electrónico. También permite conocer los contactos que tenemos actualmente guardados y editarlos. Los contactos se guardan encriptados haciendo uso del subsistema criptográfico.

3.3.4 Subsistema criptográfico

Permite almacenar la información personal del usuario, como servidores, dirección de correo electrónico, contraseña y contactos de forma segura en un archivo localizado en la misma carpeta que el programa, de manera que la carga de estos datos sea casi transparente. Se usa una contraseña maestra que el usuario define en la primera ejecución del programa para cifrar todos los datos, y que se pedirá al usuario en ejecuciones sucesivas para descifrar este archivo. Nótese que esta contraseña maestra no tiene porqué ser igual que la contraseña que se usa para acceder al email.
Es posible destruir todos los datos borrando el fichero cifrado, en cuyo caso el programa volverá a solicitar una nueva contraseña maestra, como si se tratara de una primera ejecución del programa.

3.3.5 Subsistema de autentificación

Permite al usuario acceder a su email autentificándose contra los servidores especificados por el usuario de forma segura mediante SSL.

3.3.6 Subsistema principal

El subsistema principal permite al usuario acceder a todo el conjunto de subsistemas de forma que pueda utilizar la aplicación de forma coherente y accediendo a todas las capacidades que soporta la arquitectura subyacente. Permite acceder al sistema de carpetas del email y mantener la organización de los correos en las mismas, moverse entre carpetas, cambiar un email de carpeta, acceder al subsistema de contactos, al de lectura y al de escritura desde un menú principal, aportando una funcionalidad equiparable a la de los clientes de correo electrónicos modernos actuales.

4 Desarrollo

4.1 Introducción

El proceso de desarrollo, al igual que el proceso de diseño, ha sido dividido en dos grandes fases, una para el desarrollo de la arquitectura y otra para el desarrollo del cliente de correo electrónico. Dado que una de las motivaciones del desarrollo del cliente de correo electrónico era validar la arquitectura sobre la que se construye el mismo, los procesos de validación de la arquitectura y de desarrollo del cliente de correo electrónico se superponen en el tiempo como veremos más adelante, de forma que la arquitectura ha estado en desarrollo durante todo el proyecto.

Uno de los primeros requisitos que se detectaron fue que la arquitectura debía ser multiplataforma. Es común encontrarnos que muchos programas son específicos para un único sistema operativo, un ejemplo es Microsoft Outlook, que sólo está disponible para Windows. También tenemos que no existe un lector de pantalla unificado para todos los sistemas operativos, lo que puede llevar a que software que sí es accesible para un lector de pantalla no lo sea (o lo sea en menor medida) para otro. La solución que se ha propuesto ha sido implementar completamente la arquitectura en Java, de forma que pueda ejecutarse en cualquier sistema operativo que tenga Java instalado, lo que incluye Windows, Mac y Linux.

Para aligerar el proceso de desarrollo, se ha hecho uso de un potente entorno de desarrollo integrado, IntelliJ IDEA, que ha permitido resolver dependencias gracias al sistema Gradle y generar archivos ejecutables distribuibles de forma rápida y sencilla.

4.2 Desarrollo de la arquitectura

El desarrollo de la arquitectura ha seguido un ciclo de desarrollo en cascada, es decir, después de la fase de diseño ha habido una fase de implementación, que es la que vamos a describir aquí, y posteriormente una fase de validación (realizada mediante el desarrollo del cliente de correo). Cabe destacar los requisitos que se encontraron, que, como es lógico, han moldeado todo el proceso de desarrollo.

El principal requisito que se encontró fue el de la portabilidad. Con cada vez más dispositivos con formas y sistemas operativos variados, era necesario plantear una implementación que pudiera abarcar el máximo de sistemas operativos posibles. Es por esto que la arquitectura ha sido implementada en Java, una plataforma que ha sido portada a un gran número de sistemas operativos.

Otro de los requisitos, es que dada la alta velocidad de progreso de las tecnologías de texto a voz, que cada vez suenan mejor y más realistas, es que la arquitectura pueda soportar más de un sistema de texto a voz, es decir, haya un mecanismo para añadir más sistemas TTS a la arquitectura sin necesidad de recompilar la misma.

Con el paso del tiempo y con la construcción de aplicaciones más complejas, es posible que se necesiten componentes que no hayan sido implementados en la arquitectura original, por lo tanto se proporciona un mecanismo para que sea sencillo crear nuevos componentes sin tener que recompilar la biblioteca.

Por último, cabe destacar que durante la implementación del cliente de correo electrónico, a parte de comprobar que la biblioteca ha sido lo suficientemente flexible como para soportar la mayoría de los casos de uso, en las excepciones en las que no ha cumplido las expectativas, se ha adaptado sin gran complicación de forma que fuera más potente y cumpliera con los nuevos requisitos.

4.2.1 Desarrollo general de la arquitectura

Primero, vamos a dar una breve visión general de la arquitectura de forma que más adelante podamos ir enfocándonos en cada parte más específicamente.
Vamos a presentar un diagrama de clases simplificado para demostrar la arquitectura final:

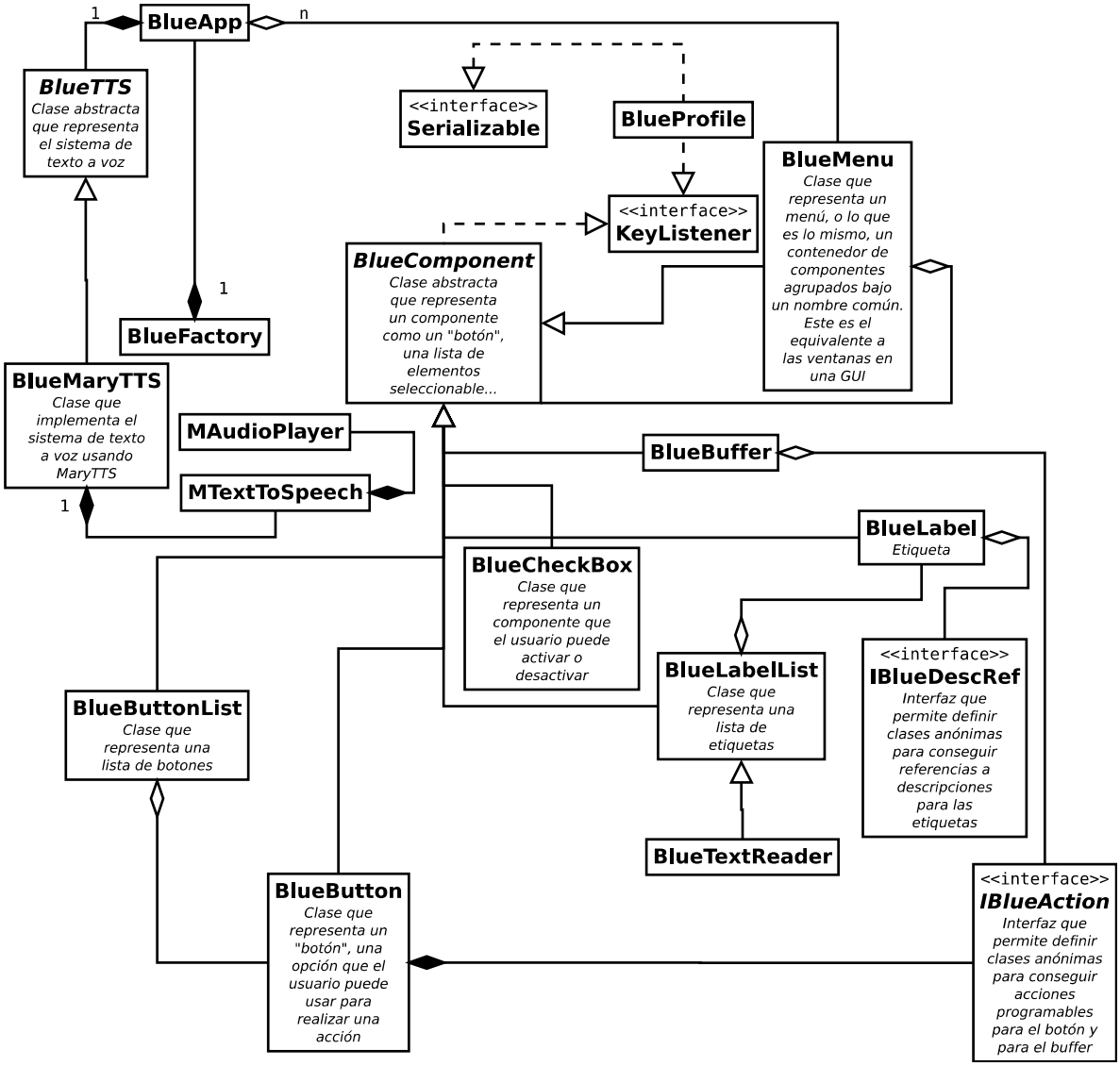

Figura 1: Diagrama de la arquitectura Blue

En Blue (el nombre que se le ha dado a la arquitectura), vamos a tener una clase principal BlueApp, que es la encargada de mantener todos los componentes ordenados y conseguir la interacción con el sistema operativo y con el sistema de texto a voz. Es la clase encargada de la ejecución efectiva del programa y de conseguir input del usuario a bajo nivel.
En concreto, se ha usado un JFrame de Swing para proporcionar leer el teclado y conseguir que los lectores de pantalla detecten la aplicación (aunque sean incapaces de leer su contenido, cosa de la que se encarga la arquitectura).

Con la clase abstracta BlueTTS logramos aislar la implementación del sistema texto a voz de la aplicación principal, de forma que sea posible extender esta clase para implementar distintos sistemas de texto a voz, como se ha hecho en este caso con MaryTTS, dando lugar a la clase BlueMaryTTS que contiene la implementación a bajo nivel de MaryTTS, desde las comunicaciones con el servidor MaryTTS hasta la implementación a bajo nivel de la

reproducción de audio (estas implementaciones a bajo nivel forman parte de código externo a este proyecto, es decir, se escapan del ámbito del trabajo). En la implementación a bajo nivel de MaryTTS se han utilizado varios módulos modificados disponibles como dominio público en internet, debido a su buena documentación, licencia y conveniencia. Estos módulos corresponden a las clases MAudioPlayer y MTextToSpeech, desarrollados por GOXR3PLUS STUDIO y con licencia Apache 2.0.

Con la clase abstracta BlueComponent conseguimos tener una base común sobre la que implementar todos los componentes, esto hace sencillo que la aplicación pueda usar nuevos componentes extendiendo esta clase. Como podemos observar, todos los componentes por defecto heredan de esta clase, que proporciona una descripción, métodos para diferenciar componentes activados (es decir, que están en menús que están activos) y componentes ocultos (que están activos pero sin embargo actúan de forma que el foco nunca se pose sobre ellos). También se dan métodos para poder implementar componentes contenedores, de forma que se pueda mover el foco entre elementos contenidos en ese componente.

Mediante el sistema de menús, con la clase BlueMenu, se ha implementado esa secuencialidad necesaria para que tenga sentido una interfaz para invidentes. Como ya hemos comentado en el capítulo de diseño, cuando los usuarios invidentes usan lectores de pantalla, navegan por los elementos de una interfaz gráfica de usuario de forma secuencial, es decir, pasamos de una interfaz gráfica bidimensional y de acceso aleatorio (porque puedes fijar la vista en cualquier elemento de la pantalla), a una interfaz sonora unidimensional y de acceso secuencial (porque oyes cada vez la descripción de un elemento y te mueves al siguiente elemento o al anterior). Este cambio de paradigma se logra a través de los menús, que son en esencia grupos secuenciales de componentes.

En el diseño aparece un sistema de perfiles, implementado en la clase BlueProfile. Este sistema está diseñado para permitir un cambio en el orden de los componentes (para permitir personalizar la interfaz de usuario en tiempo de ejecución), y se permite que su carga sea transparente para el usuario, y, por supuesto, permite guardar la modificación hecha a la interfaz mediante una combinación de teclas.

Por último, se ha creado una clase BlueFactory, que permite agregar elementos de forma sencilla a la interfaz mediante el uso de métodos (cada método añade un elemento).

4.2.2 Desarrollo del sistema texto a voz

El sistema TTS se ha simplificado de forma que sea una caja negra con tres operaciones posibles:

- Habla (*speak*): Inicia la locución de un texto.
- Habla interrumpiendo (*speakInterrupt*): Inicia la locución de un texto interrumpiendo el actual texto que se está leyendo.
- Interrumpe (*interrupt*): Para la locución actual, imponiendo silencio sobre la locución actual.

Cabe destacar la diferencia entre habla y habla interrumpiendo. Para conseguir uno de los grandes requisitos, el requisito de navegación rápida entre componentes, es necesario que el sistema de texto a voz que utilicemos soporte que se le interrumpa. Supongamos el siguiente caso hipotético:
Un usuario experimentado entra en un menú complejo con gran cantidad de componentes. El primer componente es un texto que informa de algo, y es un informe largo, que quizás

no siempre es de interés. Si el usuario tiene que escuchar todo el informe antes de poder escuchar otro segundo informe (que estaría en un componente después del primer informe), probablemente no usaría nuestra aplicación. Si el usuario desea escuchar de inmediato el segundo informe, aunque por defecto se empiece a escuchar el primer informe debe ser capaz de, al cambiar el foco al segundo, comenzar a escuchar de inmediato el segundo informe. Es por esto que se necesita capacidad de interrupción, para poder navegar rápidamente por esta secuencia de componentes. Aquí entra en juego también un diseño inteligente del programador que haga uso de la biblioteca, que debe asignar las descripciones de los elementos de forma que el usuario sea capaz de entender rápidamente qué elemento está enfocado, o al menos su función.
El lector podría preguntarse porqué no se usa la función *interrupt* seguida de un *speak* en vez de *speakInterrupt*. Esto es debido a que algunos TTS, entre los que se encuentra MaryTTS, tienen la capacidad de interrumpir más velozmente una locución con *speakInterrupt* que con *interrupt*. Esto es así porque MaryTTS por debajo inicia un servidor local al que se envían peticiones, entonces es más eficiente enviar una única petición de “di esto inmediatamente” que una de “para” seguida de otra que sea “di esto”.

Podemos observar en la figura 2 el aislamiento de la implementación a bajo nivel del sistema de texto a voz.

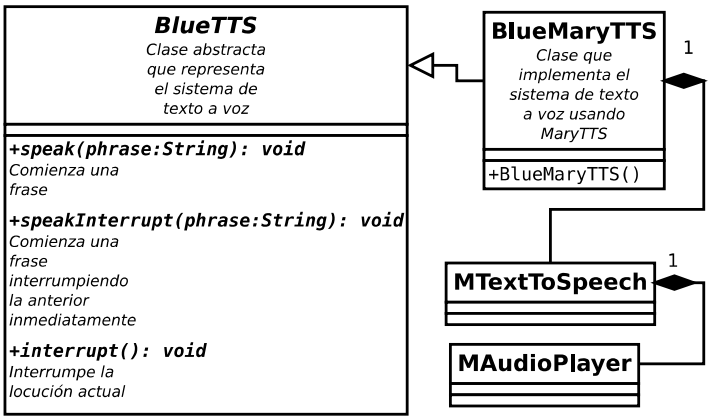

Figura 2: Diseño del sistema TTS en forma de diagrama de clases parcial

4.2.3 Desarrollo del sistema de componentes

El sistema de componentes de la arquitectura Blue tiene una clase principal denominada BlueComponent. Esta clase abstracta es de la que hereda todo componente. Entendemos por componente un elemento que tiene una descripción y cubre una funcionalidad particular.
En esta arquitectura, aunque se proporciona un componente genérico (que por cierto, no es instanciable porque en Blue un componente debe tener una funcionalidad bien definida), podemos encontrar, de forma conceptual, dos grandes grupos de componentes, los individuales y los colectivos. Un componente colectivo es aquel que contiene una secuencia de componentes (es literalmente una secuencia, tanto para el usuario como en el código). Este es el caso de los menús y de las listas, que requieren de otros componentes para funcionar. Los componentes individuales, como botones, buffers, y etiquetas tienen sentido por si solos, sin necesidad de tener asociados otros componentes.
Todo componente individual soporta que su lectura sea interrumpida por un silencio (pulsando la tecla S) o repetida (pulsando la tecla R).
En total tenemos los siguientes componentes:

- Individuales
  - Etiqueta (BlueLabel): Permite dar información estática o dinámica al usuario.
  - Botón (BlueButton): Permite realizar una acción mediante la pulsación de Enter (por defecto) u otra tecla (configurable por el programador). Para realizar la acción, es necesario que tenga el foco.
  - Búfer de entrada (BlueBuffer): El búfer de entrada es un componente análogo a un recuadro de texto en una interfaz gráfica. Se ha implementado un comportamiento modal, es decir, si el usuario pulsa control cambia entre modo descripción y modo inserción. En modo inserción el usuario puede teclear las letras de lo que desee escribir. En modo descripción puede pulsar C para copiar el contenido del búfer al portapapeles, o pulsar P para agregar al búfer contenido del portapapeles (sin borrar lo que ya hay en el búfer).
  - Checkbox (BlueCheckBox): Permite tomar decisiones binarias de forma rápida y contextualizada, presionando Enter. Para realizar la acción, es necesario que el componente tenga el foco.
- Colectivos
  - Menú (BlueMenu): Permite agrupar los componentes de forma secuencial. Es análogo a las ventanas de una interfaz gráfica. Tiene una descripción y una ayuda contextual, que puede oírse pulsando la tecla H.
  - Lista de botones (BlueButtonList): Permite agrupar botones en una lista. Por defecto, al tomar el foco una lista, toma el foco el primer botón de la lista. Se puede cambiar el foco al siguiente botón de la lista mediante la tecla de flecha hacia abajo del teclado, y al anterior con la flecha hacia arriba. El comportamiento es rotatorio, es decir, si el usuario llega al último elemento y pulsa la tecla hacia abajo, se enfoca el primer elemento de la lista.
  - Lista de etiquetas (BlueLabelList): Permite agrupar etiquetas en una lista. Por defecto su comportamiento es análogo al de una lista de botones, sólo que los elementos de la lista, al ser etiquetas, no pueden desencadenar acciones.
  - Lector de textos (BlueTextReader): Permite leer textos largos extendiendo la funcionalidad de la lista de etiquetas. El texto de este componente es subdividido en frases de forma que cada etiqueta interna contenga una frase. De esta forma es posible avanzar rápidamente por el texto o repetir una frase en particular mediante el soporte de cada etiqueta a la repetición. Cuando se enfoca la primera vez este componente, se lee el texto mediante un sistema de hilos que permite leer una frase detrás de otra sin necesidad de bloquear la aplicación. Esto tiene que ver con la forma en la que interactúa la aplicación con el TTS. Cuando la arquitectura llama al TTS, este procesa las solicitudes mediante una cola FIFO, entonces si encolamos una tras otra las frases del texto, perderemos la posibilidad de repetir instantáneamente la que se esté enfocando. Es por esto que necesitamos que esta lectura sea asíncrona y pueda haber varias colas a la vez.

Veamos todo esto en un diagrama de clases:

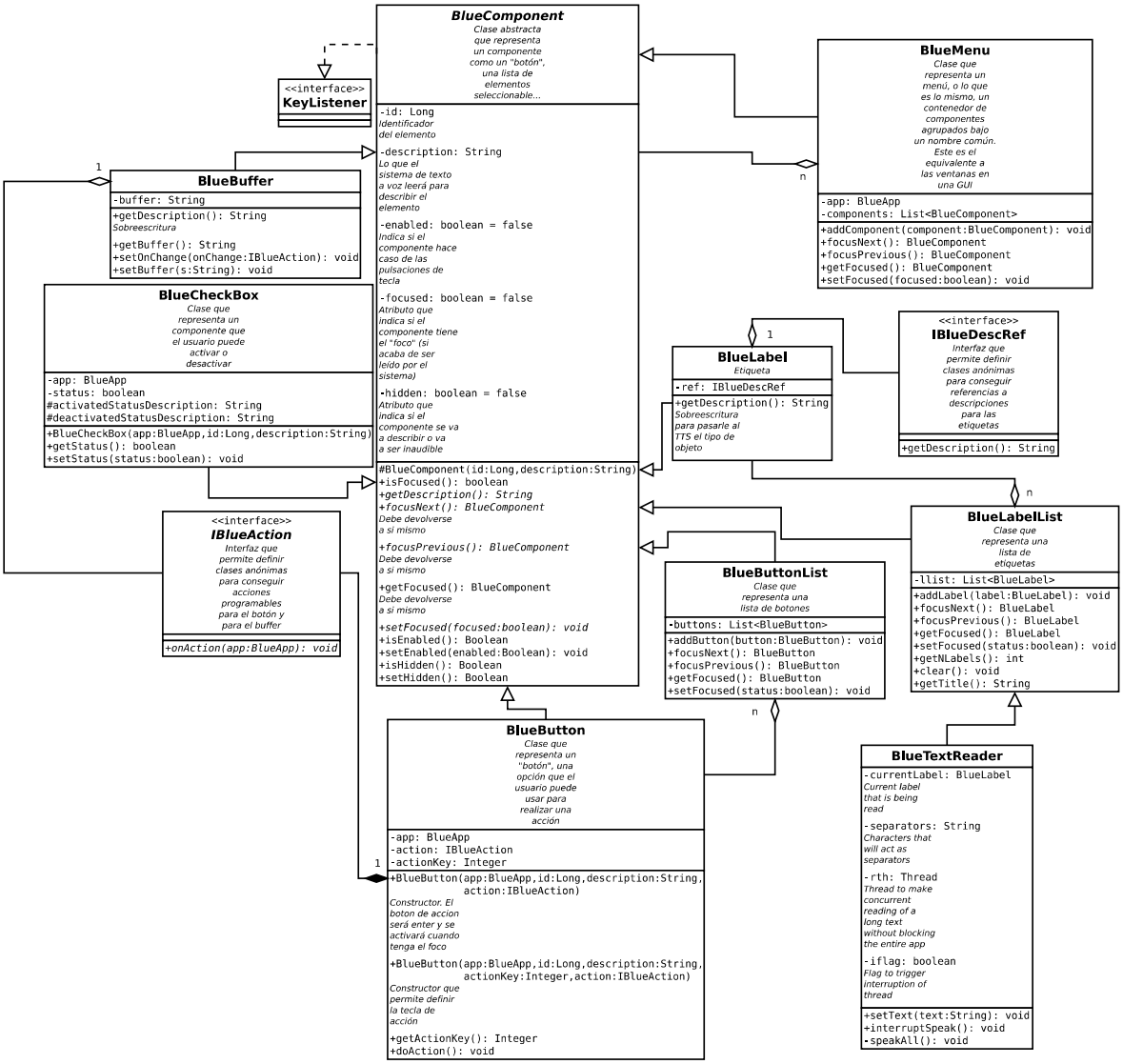

Figura 3: Diseño del sistema de componentes en forma de diagrama de clases parcial

Caben destacar también las interfaces *IBlueAction* e *IBlueDescRef*:
IBlueAction permite la creación de *callbacks* en forma de clases anónimas, por ejemplo, permite definir qué acciones se van a llevar a cabo al activar un botón mediante la sobreescritura del método *onAction* de esta interfaz.

IBlueDescRef permite obtener texto dinámico fácilmente en el componente BlueLabel sin necesidad de reasignarlo cada vez que cambie. Esto es útil a la hora de darle al usuario información que cambia con mucha frecuencia (como por ejemplo el número de mensajes no leídos en un cliente de correo electrónico).

4.2.4 Desarrollo del sistema principal

En Blue, entendemos por sistema principal el conjunto de clases compuesto por BlueApp, BlueProfile y BlueFactory. A continuación mostramos un diagrama de clases del sistema principal:

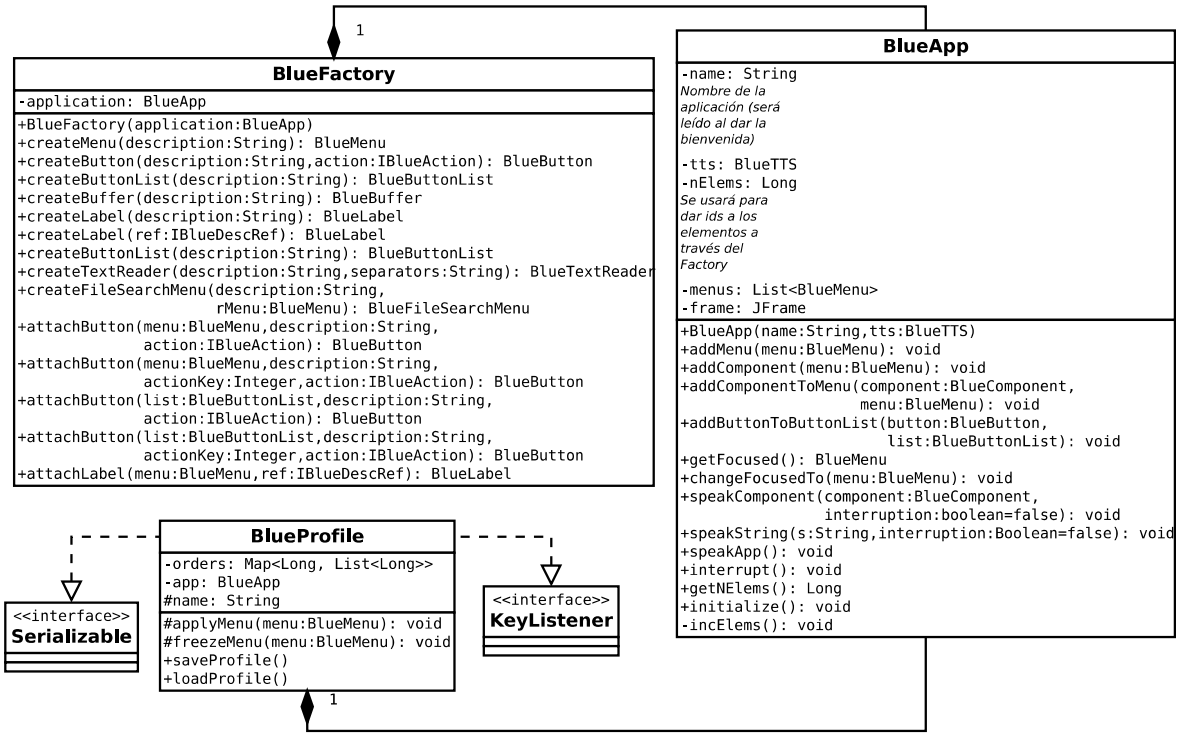

Figura 4: Diseño del sistema principal en forma de diagrama de clases parcial

BlueApp es la clase más importante del sistema, pues es la encargada de manejar los eventos de entrada y salida con el sistema operativo, y permite a los componentes acceder al sistema de texto a voz. También proporciona métodos para poder construir interfaces, es decir agregar componentes a la interfaz. Es, en definitiva, el catalizador que hace que toda la estructura tenga sentido y funcione correctamente, sirviendo como punto de interacción central con el usuario para todos los componentes.

BlueProfile es una clase estrechamente ligada a BlueApp, porque permite extender la funcionalidad de la estructura principal. Por ejemplo, esta clase sirve para implementar el cambio de orden de los componentes por parte del usuario en tiempo de ejecución, es decir, es una clase que permite la personalización de una interfaz. Creando clases que hereden de BlueProfile, se puede lograr por ejemplo crear atajos de teclado que permitan navegar por el programa de forma sencilla, o guardar información adicional del usuario, gracias a que BlueProfile implementa la interfaz *ISerializable*.

BlueFactory es una clase que permite agregar de forma sencilla componentes a una aplicación, de forma que en vez de tener que instanciar un componente y agregarlo a un menú, se puede realizar todo en un único método. Existen métodos *create* y *attach*, los métodos *create* agregan un componente al último menú añadido, y los métodos *attach* permiten agregar un componente a un menú concreto o a un componente lista.
Mediante estos métodos es posible reducir el código que hay que escribir y aumentar la claridad de los programas basados en la arquitectura.
Esta clase podría ser especialmente útil en un futuro para poder implementar de forma sencilla por ejemplo traducciones de una interfaz generada dinámicamente por medio de la biblioteca a ficheros que podrían ser leídos, y de este modo proporcionar esa misma interfaz para usuarios videntes o con otras necesidades especiales.

4.3 Desarrollo del cliente de correo electrónico

El desarrollo del cliente de correo electrónico (al que se ha denominado RedMail) ha seguido un modelo de desarrollo incremental iterativo, de forma que se pudieran evaluar las técnicas utilizadas en la arquitectura de forma controlada. La arquitectura modular de un cliente de correo electrónico también permitía este desarrollo (se pudo imaginar fácilmente que habría por ejemplo módulos diferenciados para la lectura y el envío de correos).

Se ha utilizado el patrón de diseño modelo-vista-controlador, de forma que pudiera aislarse la lógica de la interfaz, que durante el desarrollo del cliente de correo electrónico se encontraba en fase de validación, y probablemente estaría sujeta a una gran cantidad de cambios. A continuación mostramos un diagrama de clases del cliente completo:

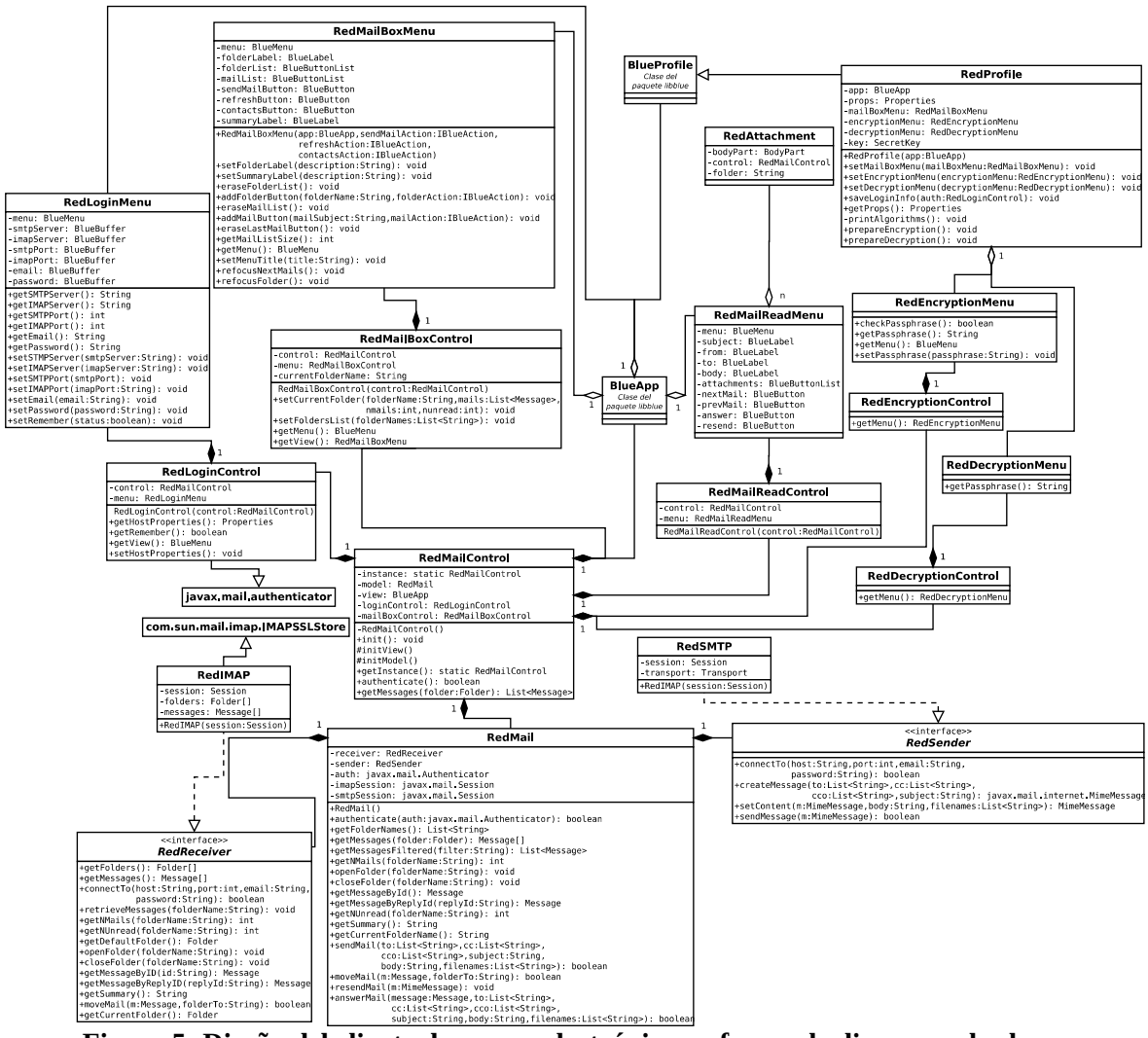

Figura 5: Diseño del cliente de correo electrónico en forma de diagrama de clases

El cliente de correo fue diseñado de forma que estuviera subdividido en varios módulos:

- El módulo de autentificación, conformado por las clases RedLoginControl y RedLoginMenu.
- El módulo criptográfico, conformado por las clases RedProfile, RedEncryptionMenu, RedDecryptionMenu, RedEncryptionControl y RedDecryptionControl.

- El módulo de contactos, conformado por las clases RedContactEditMenu, RedContactsMenu, RedContactEditControl y RedContactsControl.
- El módulo de lectura de emails, conformado por las clases RedMailReadMenu, RedMailReadControl, RedAttachment, RedReceiver y RedIMAP.
- El módulo de escritura de emails, conformado por las clases RedMailWriteMenu, RedMailWriteControl, RedSender y RedSMTP.
- El módulo principal conformado por las clases RedFolderMoveMenu, RedMailBoxMenu, RedMailBoxControl, RedMailControl y RedMail.

A continuación explicaremos cada módulo en detalle, tanto en su diseño como de cara al usuario.

4.3.1 Módulo de autentificación

El módulo de autentificación permite al usuario introducir los datos de su correo electrónico, en concreto servidores y puertos necesarios para la conexión, email y contraseña.
El funcionamiento de cara al usuario es que cuando aparece el menú de autentificación, en el caso de que no haya un perfil guardado (más adelante veremos esto), se lee el primer elemento, que es una etiqueta (BlueLabel) que indica que no hay perfil guardado (esta comprobación se realiza mediante la interfaz *IBlueDescRef* que mencionábamos antes).
Cuando el usuario pulsa el tabulador (que es la tecla de avance de foco), el foco se sitúa en el siguiente elemento, que es el *input* del servidor SMTP (BlueBuffer). Inmediatamente el sistema de texto a voz lee al usuario la descripción de ese componente.
Al estar enfocado el BlueBuffer, el usuario pulsa control para entrar en el modo de edición; acto seguido el usuario puede empezar a teclear la dirección del servidor SMTP.
Cuando acabe, pulsa control de nuevo para cambiar a modo de descripción. Se lee de nuevo la descripción de este BlueBuffer (pues sigue teniendo el foco), y también la información que ha tecleado el usuario. En concreto, se lee justo después de una breve descripción y antes de la ayuda contextual de ese componente (en la que se insta al usuario a presionar control para cambiar de modo o presionar tabulador para mover el foco). Esto es útil porque así no es necesario que el usuario lea toda la descripción, es decir puede saltarse el componente si no es de su interés o ya está relleno. Esto facilita una navegación rápida [16]. Este comportamiento es común a todos los BlueBuffer del proyecto. Cabe destacar que se va informando al usuario de cada acción que toma, por ejemplo, si cambia a modo de inserción se le notifica, si teclea algo se le dice qué tecla ha pulsado, si borra algo se le dice que letra ha borrado. Esto es adecuado para evitar que el usuario se desoriente.
Una vez rellenados los datos (recordemos que se navega de forma secuencial por la interfaz), tendrá el foco un componente BlueCheckBox, que nos permitirá decidir si queremos que estos datos recién introducidos se guarden o no. Cuando este elemento obtiene el foco, se indica el estado actual (que es que no se guardarán los datos) y se informa al usuario de que pulsando enter podrá cambiar esto al estado contrario (es decir, hará que se guarden los datos).
Hecha esta decisión, el usuario pulsará el tabulador para mover el foco al siguiente elemento, que es el botón (BlueButton) de conectar. Al pulsar enter, empezará el proceso de conexión al email, y si es correcto, el módulo de autentificación habrá acabado su trabajo y pasaría al menú principal (RedMailBoxMenu).

4.3.2 Módulo criptográfico

El módulo criptográfico actúa en varios casos de uso de la aplicación.
A la hora de realizar la configuración inicial en la primera ejecución del programa, se presenta al usuario un menú (RedEncryptionMenu) en el que se le guía a través de un pequeño tutorial de forma que pueda familiarizarse con la forma de funcionar de la arquitectura, que si bien intenta ser familiar para usuarios de lectores de pantallas, tiene algunas particularidades como por ejemplo la edición modal/dual de los búfers de entrada que pueden resultar confusas. Mediante este tutorial se guía al usuario para que pueda configurar su contraseña maestra, que se usará posteriormente para cifrar toda la información personal del usuario, como contraseñas, servidores y contactos.
Una vez realizada esta primera ejecución, en sucesivas ejecuciones se presentará al usuario un menú (RedDecryptionMenu) para que pueda descifrar sus datos mediante esa misma contraseña maestra.
El cifrado se realiza mediante el algoritmo Blowfish, debido a su velocidad y alta disponibilidad en todos los sistemas operativos soportados por la arquitectura (este algoritmo viene implementado por defecto en la máquina virtual Java). Como contrapartida no podemos almacenar con este algoritmo una alta cantidad de datos (en concreto no más de 4GB) [17], pero suponemos que al almacenarse exclusivamente texto no es probable que se llegue a alcanzar el límite, y en caso de que se alcance (porque se decida implementar almacenamiento de adjuntos offline por ejemplo) es sencillo cambiar de algoritmo extendiendo la clase RedProfile (que es la que se encarga de forma efectiva del cifrado y almacenado de los archivos, extendiendo la funcionalidad de los perfiles de la arquitectura, es decir, la clase BlueProfile).

4.3.3 Módulo de contactos

El módulo de contactos permite realizar varias tareas relacionadas con contactos.
Permite al usuario añadir nuevos contactos, es decir, dar un nombre potencialmente corto y fácil de recordar a un correo electrónico probablemente largo y difícil.
Permite al usuario editar contactos ya añadidos, por ejemplo esto puede ser útil en el caso de que un contacto cambie de correo electrónico. También es posible eliminar contactos.
Los contactos se guardan en una estructura especialmente diseñada para poder buscar rápidamente sin necesidad de escribir el contacto entero (a esto se le llama búsqueda parcial). Esta estructura se denomina árbol PATRICIA (*Practical Algorithm To Retrieve Information Coded In Alphanumeric)*, que es un caso particular del árbol radix (*radix prefix tree*). Esta estructura es altamente eficiente en tiempo de ejecución y de búsqueda parcial [18], lo que la hace particularmente apta para su uso en este caso en el que tenemos limitaciones de espacio (por el algoritmo de cifrado Blowfish) y de tiempo (porque si queremos usar búsqueda parcial, es decir, autocompletado, debemos poder recorrer la estructura muy rápidamente).

De cara al usuario, podemos acceder al módulo de contactos mediante un botón de gestión de contactos en el menú de carpetas (que veremos más adelante). Accionar este botón nos llevará a un menú en el que existe una etiqueta con el número de contactos que tenemos almacenados. Pulsando el tabulador podemos cambiar el foco a una lista de botones, representando cada botón un contacto. Si pulsamos en un contacto, podemos editarlo (se nos lleva a un menú con dos buffers de entrada, uno para el alias y otro para el email; se proporciona también un botón para guardar los cambios y para eliminar el contacto). También hay un botón para crear un nuevo contacto (por medio de un menú análogo al de edición, y de hecho se ha reutilizado), que se añadirá a la lista de contactos. Cualquier

modificación en los contactos desencadenará un proceso de guardado del perfil, es decir, el módulo criptográfico se encargará de borrar el fichero actual y cifrar y guardar de nuevo la información actualizada.
Guardar contactos tiene varios casos de uso en los módulos de lectura y de escritura, que detallaremos más adelante.

4.3.4 Módulo de lectura de emails

El módulo de lectura de emails permite leer un email. Un email es un objeto complejo que contiene una serie de datos:

- Direcciones de correo, como destinatarios, emisor y destinos en copia.
- Un contenido en forma de un objeto MIME potencialmente multiparte (es decir, puede tener archivos adjuntos) o contenido HTML embebido.
- Una serie de cabeceras que indican metadatos del email, como por ejemplo el identificador del email al que se responde en este email.

Las direcciones de correo electrónico pueden venir solas o acompañadas de un alias. En caso de que vengan acompañadas de un alias, se leerá al usuario el alias de la persona.
Si viene sin alias, se comprobará en el módulo de contactos si ese correo está guardado en el árbol. Si es el caso, se leerá al usuario el alias guardado en contactos, y si no, se leerá el correo electrónico completo.
Los adjuntos serán enumerados gracias a una lista de botones, se le da al usuario nombre y tamaño del fichero en megabytes; cuando se pulsa un botón comienza la descarga del fichero, y se avisa al usuario cuando acaba.
En el cuerpo del mensaje puede haber tanto texto plano como contenido HTML (con caracteres en negrita o cursiva por ejemplo). Como el parsing del cuerpo del mensaje es un tema muy complejo que daría para realizar otro TFG entero, se ha optado por parsear exclusivamente el texto plano descartando codificaciones extrañas y proveyendo un soporte HTML básico por medio de la biblioteca JSoup, de forma que resulte legible el texto en formato HTML. A pesar de que el parsing no es perfecto, proporciona cobertura a una gran cantidad de correos. También se ha hecho uso de expresiones regulares para eliminar el texto citado en mensajes de respuesta (Gmail por ejemplo reenvía todas las respuestas, o conversación en cada correoen forma de texto citado, lo que desorienta al usuario).
A raíz de las conversaciones (entendemos como conversación un conjunto de correos con sus respectivas respuestas), se ha implementado un sistema de lectura de conversaciones, esto es, el usuario tiene botones que puede utilizar para mostrar el padre del correo actual (es decir, el correo al que responde el correo que está leyendo actualmente), y volver al hijo. Esto se ha implementado de cero usando una versión simplificada del algoritmo REFERENCES descrito en el RFC 5256 [19]. Esto se ha implementado no sólo porque la mayoría de clientes de correo IMAP soportan esto, si no que es común en entornos corporativos el uso de conversaciones de emails en las que el usuario estaría potencialmente en copia, como por ejemplo interacciones entre distintos departamentos etc.
Otro caso de uso interesante es cuando el cuerpo contiene hipervínculos. No tiene mucho sentido leer el hipervínculo entero, que es largo y confuso. Por eso, se ha implementado un parsing de hipervínculos simplificado inspirado en el descrito en la sección 2.4.4 del estándar WCAG 2.0 [20], leyendo también algunas recomendaciones y explicaciones de WebAIM, la página de una ONG dedicada a la accesibilidad online [21] [22].
El parsing implementado consiste en sustituir en el cuerpo el hipervínculo por la cadena de texto “Link N”, siendo N el número del hipervínculo (numeramos los hipervínculos del correo por orden de aparición). Seguido al componente en el que se lee el cuerpo (que es

un BlueTextReader), hay una lista de hipervínculos (que es una lista de botones), de forma que en cada botón se lee el número de hipervínculo, parte del hipervínculo (el dominio) y se puede pulsar enter para copiar el hipervínculo completo al portapapeles.
De este modo es posible relacionar en el texto los números de hipervínculo con el hipervínculo en si de forma muy sencilla (en el estándar WCAG 2.0 se detallan mejoras a la solución que hemos implementado, que es una solución básica que demuestra que la arquitectura podría soportar soluciones más avanzadas).
A través de este menú de lectura (RedMailReadMenu), es posible acceder al módulo de escritura mediante los botones de reenvío y respuesta, acciones que veremos más adelante.

4.3.5 Módulo de escritura de emails

El módulo de escritura de emails permite escribir un email. Mediante el menú RedMailWriteMenu es posible rellenar todos los datos necesarios para enviar un email.
Primero, el usuario rellena las direcciones de correo electrónico a las que va a enviar el correo, poner en copia o copia oculta. Como hemos comentado antes, se proporciona un sistema de autocompletado. Esto consiste en una lista de botones adyacente al buffer de entrada de cada tipo de dirección (es decir, hay tres buffers y tres listas en total para destinatarios, copias y copias ocultas). Cuando el usuario empieza a escribir en un buffer, por cada carácter escrito se hace una búsqueda en el árbol de los contactos cuyo alias o correo empiece por esa cadena. Los resultados que coincidan se introducen en la lista, de forma que el usuario puede pulsar enter cuando está enfocado uno de los resultados para introducir ese correo en el buffer sin necesidad de escribirlo entero.
La escritura del texto se realiza con un buffer de entrada estándar de la biblioteca (BlueBuffer).
También es posible introducir adjuntos mediante un botón que lleva a un menú (implementado por defecto en la biblioteca) mediante el que se puede navegar por el sistema de ficheros del equipo y seleccionar un archivo.
El menú ha sido implementado de tal manera que pueda ser usado tanto como para escribir emails desde cero como para responder y reenviar emails; esto se ha conseguido rellenando los campos adecuados en cada caso:

- En el caso de responder, se rellena la dirección de destino y el asunto del menú con los valores adecuados.
- En el caso de reenviar, se rellena el asunto con una marca delante que indica que es un mensaje reenviado. Se oculta la lista de adjuntos y el botón de agregar adjuntos.

En el caso de las respuestas, el cliente de correo de forma interna se asegura de que las cabeceras estén adecuadamente rellenas para que el email quede en forma de conversación conforme al RFC 5256, y sea visto como una conversación por todo cliente que lo soporte (en este caso si que se cumple totalmente la especificación).

4.3.6 Módulo principal

El módulo principal lo componen un conjunto de menús (con sus respectivos controladores, al igual que el resto de módulos) que permiten realizar las siguientes tareas:

- Acceder a las carpetas IMAP y listar los correos contenidos en cada una de ellas, de forma que pulsando enter al estar sobre uno de los correos (cada correo es un BlueButton) lleve al usuario al menú de lectura (rellenando los componentes con los datos de ese correo en particular), proporcionando así un acceso al módulo de lectura. Este acceso se hace en el menú “principal” o de carpetas (RedMailBoxMenu).

- Acceder al módulo de contactos por medio de un botón en ese mismo menú principal.
- Acceder al módulo de escritura de emails mediante un botón que permite al usuario escribir un nuevo email, llevando al usuario al menú de escritura de emails (RedMailWriteMenu).
- Mover emails de carpeta al pulsar M estando enfocado un correo concreto, llevando al usuario a un pequeño menú con una lista de carpetas en el que el usuario puede seleccionar la carpeta a la que se va a mover el email (RedFolderMoveMenu).
- Buscar emails mediante un buffer en el propio menú principal y un botón, que en caso de estar vacío el buffer, el botón cambia de comportamiento dinámicamente y recarga los correos de la carpeta actual (indicando en la descripción del componente botón la funcionalidad actualizada).
- Proporcionar atajos de teclado para desplazarse rápidamente por la interfaz mediante la extensión del sistema de perfiles (RedProfile), como por ejemplo, pulsar N (de New) para escribir un nuevo correo electrónico, o pulsar I (de Inbox) para volver al menú principal.

4.3.7 Diagrama de navegación

Una vez entendido el funcionamiento de la arquitectura y del cliente de correo, vamos a ver un diagrama de navegación completo de RedMail, que incluye los menús y los componentes asociados a los mismos.

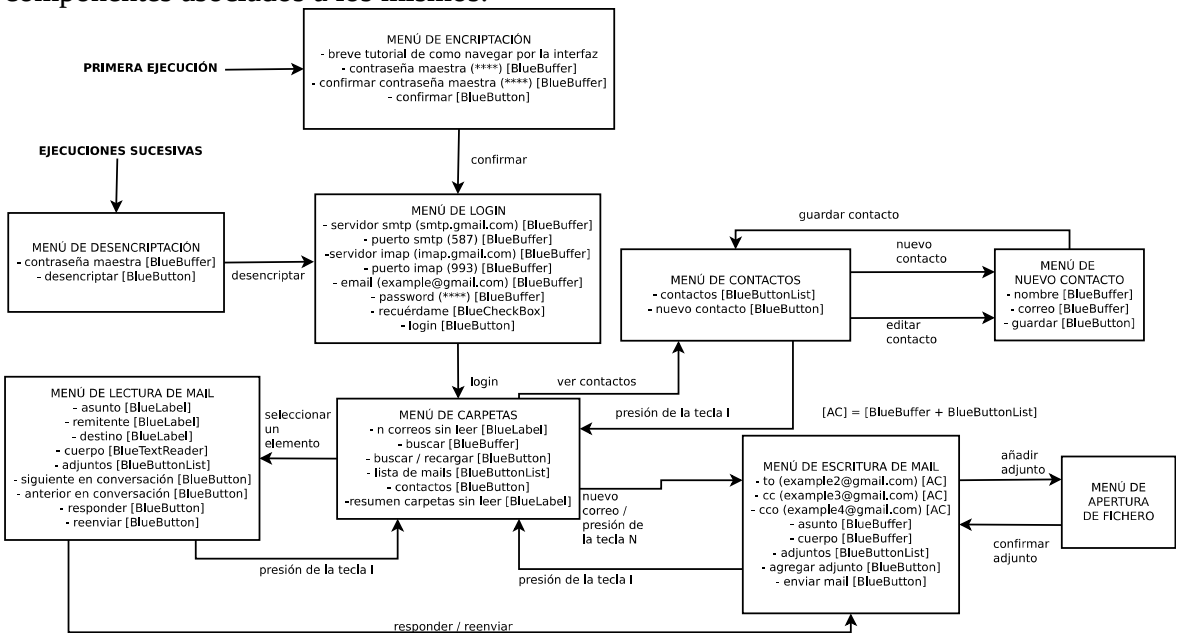

Figura 6: Diagrama de navegación del cliente de correo RedMail

Como podemos comprobar, la cantidad de menús necesarios para implementar la aplicación ha sido bastante moderada, mediante nueve menús se ha podido proporcionar la mayoría de la funcionalidad presente en los clientes de correo electrónicos modernos.

5 Integración, pruebas y resultados

5.1 Integración con el patrón modelo-vista-controlador

El cliente de correo electrónico que se ha usado para validar la arquitectura es un programa complejo que ha sido creado con la intención explícita de mejorar la biblioteca presentada anteriormente. Es por esto que se ha decidido usar un popular patrón de diseño de aplicaciones con interfaz, el patrón modelo-vista-controlador, para validar que la arquitectura es compatible con este patrón de diseño.

De este modo, tenemos que las clases de la arquitectura toman distintos roles, es decir, no todo actúa como vista.

Teniendo esto en mente, el diseño de la arquitectura se hizo de tal manera que, en las aplicaciones que se crearan con la arquitectura:

1. Fueran específicamente vista el sistema de componentes (sin contar las interfaces *IBlueAction* e *IBlueDescRef*) y la clase BlueFactory del sistema principal. Podemos ver esto en el encapsulamiento de cada componente menú de la arquitectura y sus componentes asociados en una única clase que conforma un menú de aplicación.
2. Fueran específicamente controlador el resto de clases.
3. No hubiera modelo proporcionado en la biblioteca, cosa que tiene sentido porque en el patrón modelo-vista-controlador el modelo es la lógica de cada programa en particular, en el caso del cliente de correo electrónico conforman el modelo las clases RedMail, RedIMAP y RedSMTP.

Cabe destacar el caso de la clase RedAttachment, que implementa la interfaz *IBlueAction*, de forma que puede almacenar más información dentro de la propia clase que va a ejecutar la acción, aparte de poder predefinir acciones para esta información variable, que podemos entender como parámetros (de hecho son parámetros en el constructor, y como un RedAttachment es una callback, son en efecto parámetros de la callback). Esto es un claro ejemplo de controlador cuya base ha sido implementada en la arquitectura.

5.2 El proceso de desarrollo incremental iterativo

Para la creación del cliente de correo electrónico se ha seguido un proceso de desarrollo incremental iterativo con tres iteraciones. A continuación vamos a exponer qué se ha hecho en cada iteración.

5.2.1 Iteración 1. Módulos de autentificación, principal y de lectura

Durante la primera etapa del desarrollo de RedMail, el primer paso fue implementar la estructura base siguiendo el patrón modelo-vista-controlador, de forma que fuera posible realizar todo el desarrollo siguiendo este modelo. Para hacer esto, se empezó implementando el módulo de autentificación completo, es decir, menú, controlador y modelo. En esta fase aún no había un componente BlueCheckBox implementado en la arquitectura, y fue durante el desarrollo del módulo de autentificación cuando se implementó este componente dentro de la biblioteca libblue.

Después, se comenzó el desarrollo del menú principal, y hubo una toma de contacto con la biblioteca JavaMail, usada para implementar la comunicación con los servidores de correo. Dado el soporte SSL de esta biblioteca, se decidió agregar esta seguridad a las comunicaciones del cliente de correo (disponibles también en los clientes de correo electrónicos modernos). En la toma de contacto con el módulo IMAP de JavaMail, dada la estructura de carpetas del protocolo (los correos se guardan en carpetas, que por ejemplo Gmail denomina etiquetas) se decidió unificar el menú de recibidos y enviados (se había

pensado así en un principio) en un mismo menú que pudiera abarcar todas las carpetas, es decir el menú principal que hemos descrito antes, RedMailBoxMenu.
Se implementó el menú de lectura, en el que se pudo notar por primera vez el problema de la lectura de un texto largo, tanto en términos de usabilidad como de rendimiento (el TTS tardaba un tiempo en leer el componente que contenía el cuerpo, que en su momento era un BlueLabel, es decir, una etiqueta simple).
Fue en este momento cuando se decidió diseñar un componente especial para la lectura de textos largos, que sería implementado en iteraciones posteriores (dando lugar a la clase BlueTextReader).
Se implementa también una visión de conversación preliminar con un algoritmo aún más simplificado que el final, con la contrapartida de ser extremadamente lento.
Se detecta el problema del texto citado por Gmail en conversaciones.
Se decide dejar la gestión de archivos adjuntos, prevista para esta iteración, para la siguiente iteración.
Se detecta la necesidad de poder descubrir en tiempo de ejecución la posición del foco, es decir, que pulsando una tecla (en nuestro caso será W de where, dónde) se diga al usuario en qué menú está y qué componente está enfocado, y se implementa esta funcionalidad en la biblioteca. También se detecta la necesidad de que el usuario pueda silenciar el sintetizador de voz en cualquier momento, se implementa de tal modo que pulsando S el TTS se calle [23].
Se agrega el atajo de teclado en el cliente de correo electrónico para moverse al menú principal (tecla I).

5.2.2 Iteración 2. Módulos de contactos, encriptación y escritura

Durante esta iteración, se crean los módulos de contacto, encriptación y escritura.
Dada la complejidad de la estructura PATRICIA para el sistema de contactos, se reutiliza una implementación libre de Apache Commons.
Se implementó el módulo de encriptación, adaptando la forma en la que se guardaban los datos en la clase RedProfile; se hace uso del módulo criptográfico de Java.
Se produce una toma de contacto con el módulo SMTP de JavaMail, que si bien cuadra bastante bien en la estructura modelo-vista-controlador que teníamos, hubo que realizar cambios menores en la manera en la que se guardaba en memoria los datos del servidor (pues ya no era una dirección de servidor sino dos, el servidor IMAP y el servidor SMTP).
Se detecta la necesidad de incluir eventos en el componente BlueBuffer de la biblioteca, así que se implementa incluyendo un atributo *IBlueAction* dentro de la clase, de tal forma que se pueda ejecutar una acción cada vez que el buffer cambia (es decir, cuando el usuario añade o borra un carácter). Esto resulta necesario para implementar el autocompletado.
Se implementa, fruto de la validación de la iteración anterior, la funcionalidad de descubrimiento presionando W y el silencio pulsando S, el componente BlueTextReader, la descarga de adjuntos, el parsing del texto citado por Gmail y diversas mejores contextuales en el cliente de correo (aclaraciones en las descripciones de los componentes etc).
Se detecta un fallo en la máquina virtual Java de Windows, que impide al BlueBuffer procesar una arroba en teclados españoles y alemanes con el método actual de gestión de eventos de teclado de la biblioteca [24]. La solución de momento para los usuarios de Windows es escribir su correo en otro sitio y copiarlo y pegarlo en el buffer o haber almacenado previamente la información como el correo del usuario o de los contactos en un sistema Linux y llevarlo en una memoria USB.

5.2.3 Iteración 3. Mejoras en el módulo principal y de encriptación e interconexión de los módulos de lectura y escritura

Durante esta iteración se realizan mejoras en el módulo principal y se agregan las funcionalidades de reenvío y respuesta, interconectando de este modo los módulos de lectura y escritura. Se añade también un pequeño tutorial interactivo.
Se agrega al menú principal la funcionalidad de búsqueda, soportada en la biblioteca JavaMail. Se encuentra un bug en JavaMail por el que la búsqueda IMAP toma un largo tiempo; se intenta solventar este bug limitando el espacio de búsqueda (haciendo que tenga que haber un mínimo de palabras escritas en el buffer de búsqueda), pero aún así el tiempo que se tarda en buscar un correo electrónico llega hasta los dos minutos. Se intentan varias soluciones encontradas en internet sin éxito, por lo que se deja para trabajo futuro arreglar la baja velocidad de búsqueda de correos.
Se añade un tutorial interactivo en el módulo de encriptación para la primera vez que se ejecuta el programa. Este tutorial intenta que el usuario tenga una primera toma de contacto con la interfaz de forma amigable y guiada, explicando al usuario el funcionamiento de la interfaz paso a paso y terminando con la configuración de la contraseña maestra que se usará para cifrar los datos.
Se implementa la ayuda contextual en los menús de la arquitectura, fruto de la validación de la iteración anterior. Pulsando la tecla H (de help), el usuario puede escuchar una descripción de cuales son las acciones posibles en el menú en el que se encuentra.
Se comienza una refactorización de gran parte de la biblioteca de forma que sea posible usar cualquier lenguaje (actualmente sólo soporta inglés) para describir cada elemento. Se consigue llevar a cabo toda la refactorización en la biblioteca a excepción de la clase BlueProfile, que se deja para trabajo futuro.
Se descubre y corrige un bug en la implementación del cliente de correo por el que se produce una desconexión del servidor SMTP después de un tiempo de uso prolongado.
Se mejoran las descripciones de los elementos y la contextualización en general de la aplicación.
Fruto de la validación de esta iteración se mejora el tutorial y se siguen mejorando las descripciones de los elementos y la ayuda contextual.

5.3 Pruebas con usuarios

Debido a la complejidad del proyecto, las pruebas con usuarios no fueron las que hubieran sido deseables. Por cada iteración se fueron comprobando con el tutor que la interfaz resultaba intuitiva, y dada la amplia experiencia del tutor en materia de accesibilidad se pudieron realizar una gran cantidad de cambios en las descripciones de los elementos y la contextualización de la aplicación.
Más allá de esta validación iterativa, se realizó una prueba final de usabilidad con un usuario que, si bien no era invidente, al no tener interfaz visual el programa, la visibilidad del sujeto no supondría problema alguno. Por supuesto, hay que tener en cuenta que probablemente un usuario invidente estaría más acostumbrado a utilizar una interfaz más parecida a la que proporciona nuestra arquitectura, y por lo tanto su desempeño en las pruebas sería mejor. Para compensar la desorientación del usuario vidente, yo realizo la misma prueba y expongo los resultados de mi propio desempeño en la prueba en este trabajo, de forma que pueda verse el “ideal” de desempeño, es decir el máximo desempeño teórico (y recalco lo de teórico, luego vemos el porqué) que se puede alcanzar en la prueba.
En total se cronometran diecinueve pruebas, que conforman el recorrido completo por la interfaz del cliente de correo electrónico.
Las pruebas están normalizadas, es decir se usan siempre los mismos correos electrónicos para las pruebas, con las mismas carpetas y correos en cada una.

5.3.1 Usuario 1

El usuario 1 es vidente con un nivel bajo-medio de inglés y acostumbrado a usar en el día a día un cliente IMAP (Microsoft Outlook).

Nº PRUEBA	DESCRIPCIÓN	PRECONDICIONES	PASA	TIEMPO EMPLEADO (m:ss)	COMENTARIOS
1	Tutorial	-	SI	5:52	Se le dicta al usuario una contraseña maestra para utilizar
2	Login	Pasado tutorial	SI	4:48	Se le dicta al usuario los puertos y servidores para mayor fluidez
3	Cambio de carpeta	Estar logueado	SI	1:14	
4	Identificar un email concreto	Estar en carpeta	SI	0:10	
5	Identificar emisor de un email	Haber abierto email	SI	0:05	
6	Identificar cuerpo de un email	Haber abierto email	SI	0:05	
7	Identificar hipervínculos de un email	Haber abierto email	SI	0:16	
8	Identificar adjuntos de un email	Haber abierto email	SI	0:15	
9	Responder email	Haber abierto email	SI	1:12	
10	Adjuntar archivo	Haber abierto email	SI	2:00	
11	Volver a inbox desde escritura	Haber escrito email	SI	0:02	
12	Agregar contacto	Estar logueado	SI	3:03	
13	Volver a inbox desde contactos	Estar en contactos	SI	0:01	
14	Escribir email	Estar logueado	SI	1:32	
15	Usar autocompletado	Estar escribiendo email	SI	0:10	
16	Editar un contacto	Estar logueado	SI	0:43	El usuario se queja de que no puede parar la ayuda contextual
17	Refrescar la carpeta actual	Estar logueado	SI	0:50	
18	Cambiar un email de carpeta	Estar logueado	SI	0:25	
19	Buscar un email usando el buscador	Estar logueado	SI	0:42	Sin contar el tiempo de carga debido a la biblioteca IMAP
			TOTAL	23:25	Tiempo cronometrado del recorrido completo

Tabla 1: Desempeño en pruebas del usuario 1

Los comentarios más destacables del usuario han sido que el sistema texto a voz era complicado de entender, que resulta bastante liosa la escritura modal, es decir que se use

control para cambiar de modo de edición y sin embargo se use enter para activar botones, y encima con el tabulador se cambia de componente, al usuario le resulta complicado manejarse con la interfaz.
El usuario reconoce que necesita un periodo de adaptación, porque en las últimas pruebas empezaba a sentirse más cómodo con el programa, y se notaba bastante la mejora en la fluidez del usuario.

Observando al usuario, se ha podido discernir lo siguiente:

- El usuario duda bastante entre pulsar control o enter en cada input buffer.
- El usuario usa la ayuda contextual cuando pierde el foco (tanto W como H) y lo encuentra muy útil, pero aún así parecía esperar algo más de las ayudas contextuales.
- Consigue un gran desempeño a la hora de cambiar emails de carpeta.
- El usuario ha acabado cansado después de las pruebas. La voz del TTS le resultaba estresante, el usuario habría estado mucho más cómodo si el idioma hubiera sido nativo. Achacamos esto a la desorientación propia de usar tecnología accesible no estando acostumbrado a ella (era la primera vez que utilizaba tecnología de este tipo), al idioma y a que el propio TTS no vocaliza bien.

En definitiva, el usuario consigue usar el cliente de correo con autonomía, equivocándose de vez en cuando debido a la inexperiencia y a la mala comprensión del sistema de texto a voz, pero demuestra que le resulta posible usar el programa y que es medianamente intuitivo.

5.3.2 Usuario 2

El usuario 2 es el desarrollador tanto del programa como de la arquitectura. Para realizar las pruebas con mayor fidelidad, se realizaron las pruebas con dos semanas de aislamiento tanto de la arquitectura como del cliente de correo, de forma que el usuario no recordara el orden exacto de los componentes en cada menú y tuviera que prestar cierta atención a las descripciones. Esto representa el tiempo mínimo posible en completar cada prueba.

Nº PRUEBA	DESCRIPCIÓN	PRECONDICIONES	PASA	TIEMPO EMPLEADO (m:ss)	COMENTARIOS
1	Tutorial	-	SI	2:24	
2	Login	Pasado tutorial	SI	1:33	0:05 cuando están recordados los datos
3	Cambio de carpeta	Estar logueado	SI	0:25	
4	Identificar un email concreto	Estar en carpeta	SI	0:20	
5	Identificar emisor de un email	Haber abierto email	SI	0:05	
6	Identificar cuerpo de un email	Haber abierto email	SI	0:10	
7	Identificar hipervínculos de un email	Haber abierto email	SI	0:08	Aproximadamente por cada hipervínculo
8	Identificar adjuntos de un email	Haber abierto email	SI	0:02	Aproximadamente por cada adjunto
9	Responder email	Haber abierto email	SI	0:59	Añadiendo un mensaje genérico al input del cuerpo
10	Adjuntar archivo	Haber abierto email	SI	1:02	Añadiendo un adjunto en la misma carpeta (no hay navegación por carpetas)
11	Volver a inbox desde escritura	Haber escrito email	SI	0:00	
12	Agregar contacto	Estar logueado	SI	1:06	
13	Volver a inbox desde contactos	Estar en contactos	SI	0:00	
14	Escribir email	Estar logueado	SI	1:24	
15	Usar autocompletado	Estar escribiendo email	SI	0:05	
16	Editar un contacto	Estar logueado	SI	0:30	
17	Refrescar la carpeta actual	Estar logueado	SI	1:07	
18	Cambiar un email de carpeta	Estar logueado	SI	0:42	
19	Buscar un email usando el buscador	Estar logueado	SI	0:40	0:40 usando la interfaz, 4:12 por el uso de la biblioteca IMAP
			TOTAL	12:42	Tiempo cronometrado del recorrido completo

Tabla 2: Desempeño en pruebas del usuario 2

5.3.3 Comparación de resultados del usuario 1 frente al usuario 2

Para comprender mejor cuanto ha podido aprovechar el usuario 1 el potencial brindado por la arquitectura, presentamos la siguiente tabla:

Nº PRUEBA	DESCRIPCIÓN	TIEMPO USUARIO 1	TIEMPO USUARIO 2	FACTOR DE APROVECHAMIENTO
1	Tutorial	352	144	40,91%
2	Login	288	93	32,29%
3	Cambio de carpeta	74	25	33,78%
4	Identificar un email concreto	10	20	200%
5	Identificar emisor de un email	5	5	100%
6	Identificar cuerpo de un email	5	10	200%
7	Identificar hipervínculos de un email	16	8	50%
8	Identificar adjuntos de un email	15	2	13,33%
9	Responder email	72	59	81,94%
10	Adjuntar archivo	120	62	51,67%
11	Volver a inbox desde escritura	2	0	0%
12	Agregar contacto	183	66	36,07%
13	Volver a inbox desde contactos	1	0	0%
14	Escribir email	92	84	91,3%
15	Usar autocompletado	10	5	50%
16	Editar un contacto	43	30	69,77%
17	Refrescar la carpeta actual	50	67	134%
18	Cambiar un email de carpeta	25	42	168%
19	Buscar un email usando el buscador	42	40	95,24%

Tabla 3: Desempeño del usuario 2 frente al usuario 1

El factor de aprovechamiento es el incremento de velocidad del usuario 1 con respecto al usuario 2, que es el ideal, por lo tanto es lo mismo que ver cómo de cerca está el desempeño del usuario de pruebas del desempeño ideal.
Esto nos permite ver si el usuario ha tenido un desempeño ideal (factor de aprovechamiento 100%), por encima del ideal (factor de aprovechamiento > 100%), o por debajo del ideal (factor de aprovechamiento < 100%). Vamos a considerar que el desempeño es bueno cuando el factor de aprovechamiento sea mayor del 50% (suponemos que un usuario tarda de media el doble que el programador en realizar las tareas, y de hecho, en total, podemos observar que lo hace).
Lo que podemos observar de esto es que el usuario de pruebas ha realizado mejor de los esperado las tareas más simples, como las 4, 5, 6, 17 y 18.
Ha tenido un desempeño bastante bueno en tareas relacionadas con la escritura de correos como la 9 y la 14.
El autocompletado en la tarea 15 no ha resultado del todo satisfactorio para el usuario.

El sistema de contactos ha resultado complicado para este usuario, pero con las modificaciones realizadas a raíz de esta prueba se espera que en futuras pruebas haya mejores resultados.
Se debe pulir el algoritmo de parsing de los hipervínculos, y la contextualización del tutorial y del menú de login. Estos malos resultados en el tutorial y el menú de login pueden estar debidos a que han sido las primeras pruebas y el usuario todavía estaba desconcertado sobre como utilizar la interfaz.
En general, la navegación en el menú principal tiende a ser confusa, probablemente porque es el primer menú complejo con el que se encuentra el usuario. Esto parece ser así porque en la prueba 17, en la que se usa este menú, el resultado ha sido mejor que el ideal.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo de fin de grado se presenta una arquitectura mediante la que es posible desarrollar de forma fácil interfaces de usuario para invidentes, cumpliendo ciertos estándares de accesibilidad y permitiendo dar una mayor personalización y contextualización que los lectores de pantalla actuales.

Mediante esta arquitectura es posible implementar interfaces complejas, permitiendo extender fácilmente la funcionalidad estándar de la arquitectura para conseguir comportamientos que son difíciles de lograr con lectores de pantalla.

Se proporciona una experiencia de usuario que resulte amigable para usuarios de lectores de pantalla, imitando la navegación entre componentes.

La arquitectura es altamente modular, por lo que permite agregar nuevos sistemas de texto a la arquitectura sin necesidad de recompilarla, permite agregar fácilmente nuevos componentes que puedan aportar nueva funcionalidad, y permite crear fácilmente nuevos atajos que ayuden al usuario a navegar por la interfaz.

Todo esto ha sido validado mediante la implementación de un cliente de correo electrónico usando la arquitectura. Este programa incluye la mayoría de las funcionalidades esperables en un cliente de correo moderno, entre las que se encuentran capacidad de leer y enviar correos con adjuntos de forma segura (usando SSL), interpretar información HTML, guardar contactos y buscar correos.

Mediante el proceso incremental iterativo se ha ido mejorando la arquitectura de forma que fuera más robusta y potente, y a falta de usuarios de prueba para validar cada iteración, se han consultado diversos estándares que abalan las decisiones tomadas, e intentando que la arquitectura sea lo suficientemente flexible como para poder cumplir algunas características de estos estándares.

Por último, se hizo una prueba con un usuario vidente para intentar descubrir los puntos fuertes y débiles del cliente de correo y de la arquitectura, y se llegó a la conclusión de que la arquitectura es intuitiva pero requiere un tiempo de adaptación y cierta pericia por parte del programador que la use, en tanto que debe ser capaz de agregar las descripciones adecuadas a los componentes e incluir las ayudas contextuales necesarias para que el usuario no se pierda (o al menos pueda reubicarse rápidamente).

También se ha enviado un artículo corto al congreso Interacción 2019, con el resultado de haber sido aceptado. Se puede encontrar el artículo completo como anexo de esta memoria.

En definitiva, se ha llegado a implementar una arquitectura para la programación de interfaces de usuario específicamente diseñadas para invidentes, y se trata de una arquitectura portable, potente, funcional y abalada por estándares de accesibilidad y por un programa completo y útil que demuestra que es posible crear interfaces complejas con ella.

6.2 Trabajo futuro

Dada la naturaleza modular de la arquitectura, el trabajo futuro es infinito.

Los principales esfuerzos deberían centrarse en hacer que la arquitectura sea multilenguaje, porque de momento sólo soporta sistemas de texto a voz en inglés. Cuando esto esté conseguido, debería intentarse agregar mayor soporte a lenguajes, o bien añadir el lenguaje español a MaryTTS, o bien agregar más TTS a la arquitectura. Una buena opción podría ser añadir soporte para eSpeak. También sería interesante implementar un sistema de descubrimiento en tiempo real de lectores de pantalla. La arquitectura podría descubrir si hay algún lector de pantalla ejecutándose en el sistema y usar el TTS del lector de pantalla, que idealmente estaría configurado por el usuario de forma que la voz fuera óptima para el usuario.

Otra posibilidad sería crear un lector de pantallas multiplataforma y portable con esta arquitectura. No parece muy descabellado usar técnicas de reconocimiento de patrones para descubrir objetos complejos en la pantalla y traducir esos objetos a componentes de la arquitectura. Esto abre un nuevo abanico de posibilidades, por ejemplo en el ámbito del juego online (que tiende a no ser accesible para invidentes por motivos obvios), en el que ya se han aplicado con éxito técnicas de reconocimiento de imágenes [25].

También podría ser interesante portar la arquitectura a sistemas móviles, como puede ser Android. Esto es viable dado que la arquitectura está programada en Java. Sería necesario remodelar un poco la forma en la que se procesan los eventos, pero puede hacerse.

Con respecto al cliente de correo, también es posible mejorarlo. El email es una tecnología que ha tenido un largo recorrido y que no parece que vaya a abandonarnos pronto. Aunque el objetivo principal del cliente de correo en este trabajo ha sido el de validar la arquitectura, no podemos olvidar que es un programa completo y funcional que puede extenderse y al que se le pueden agregar una gran cantidad de funcionalidades, como por ejemplo una mejor gestión del parsing de hipervínculos, mejoras en el sistema de búsqueda de emails, y por supuesto, puede seguir sirviendo para validar futuros cambios en la arquitectura.

En definitiva, el trabajo futuro en relación a este trabajo es tanto como los avances tecnológicos inaccesibles que vayan surgiendo, que, dada la velocidad a la que funciona el mundo y en el que el objetivo primordial es producir para la mayoría, será muy numeroso.

Referencias

[1] “Definición de Ingeniería”, 2009, Julián Pérez Porto y María Merino https://definicion.de/ingenieria/

[2] “Spotify is not accesible for the blind”, 2016, Spotify Community https://community.spotify.com/t5/Closed-Ideas/Windows-and-Web-application-Spotify-please-make-it-accessible/idi-p/1472882

[3] “Screen Reader User Survey #7 Results”, 2017, WebAIM https://webaim.org/projects/screenreadersurvey7/

[4] “NV Access”, 2019, https://www.nvaccess.org/

[5] “Accesibilidad visual”, 2019, Apple https://www.apple.com/es/accessibility/mac/vision/

[6] “Bienvenido a Orca”, 2019, GNOME Help https://help.gnome.org/users/orca/3.26/introduction.html.es

[7] “Cloud Text-to-Speech API”, 2019, Google Cloud https://cloud.google.com/text-to-speech/pricing

[8] “Cognitive Services Directory”, 2019, Microsoft Azure https://azure.microsoft.com/en-us/services/cognitive-services/directory/speech/

[9] “Festvox: Festival”, 2019, Alan W Black http://www.festvox.org/festival/

[10] “eSpeak: Speech Synthesizer”, 2019, eSpeak http://espeak.sourceforge.net/

[11] “The MARY Text-to-Speech System (MaryTTS), 2019, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) http://mary.dfki.de/

[12] “Email is not dead. But email is chainging”, 2019, https://www.emailisnotdead.com/

[13] “Accessible Email Client”, 2012, Google Groups https://groups.google.com/forum/#!topic/alt.comp.blind-users/mxTRBOOAgXM

[14] “The best email service provider of 2019”, 2019, TechRadar https://www.techradar.com/news/best-email-provider

[15] “Usability Evaluation of Email Applications by Blind Users”, 2011, Brian Wentz, Jonathan Lazar

[16] “The Tools of a Blind Programmer”, 2016, Parham Doustdar https://www.parhamdoustdar.com/2016/04/03/tools-of-blind-programmer/

[17] “GnuPG Frequently Asked Questions”, 2019, Robert J. Hansen, A.M. Kuchling https://gnupg.org/faq/gnupg-faq.html#define_fish

[18] “PATRICIA – Practical Algorithm To Retrieve Information Coded In Alphanumeric”, 1968, Donald R. Morrison, Journal of the ACM, Volume 15 Issue 4

[19] “RFC 5256, Internet Message Access Protocol – SORT and THREAD Extensions”, 2008, M. Crispin, K. Murchison, Carnegie Mellon University https://tools.ietf.org/html/rfc5256

[20] “WCAG 2.0”, 2016, W3C, https://tools.ietf.org/html/rfc525

[21] “Introduction to Links and Hypertext”, 2019, WebAIM, https://webaim.org/techniques/hypertext/

[22] “Designing for Screen Reader Compatibility”, 2019, WebAIM, https://webaim.org/techniques/screenreader/

[23] “ETSI EN 301 549 v1.1.2 (2015 – 04), página 86, sección C.5.1.3.4, Speech output user control”, 2015, European Telecommunications Standards Institute

[24] "Bugzilla – Bug 280562 – support for alt+gr keys on keyboards that have the key", 2009, Ketan Padegaonkar, Pascal Gélinas, Eclipse Foundation https://bugs.eclipse.org/bugs/show_bug.cgi?id=280562
[25] "Detección y reconocimiento de elementos en mesas de póker online", 2018, Manuel Gómez Campo, Universidad Autónoma de Madrid

Glosario

API	Application Programming Interface
TTS	Text To Speech

Anexos

6.3 Artículo enviado a Interacción 2019

Arquitectura para el desarrollo de interfaces de correo electrónico para invidentes

David Doyáguez
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Madrid, España
david.doyaguez@estudiante.uam.es

Javier Gómez Escribano
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Madrid, España
jg.escribano@uam.es

Germán Montoro
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Madrid, España
german.montoro@uam.es

Juan Carlos Torrado
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Madrid, España
juan.torrado@uam.es

ABSTRACT

En este artículo presentamos una arquitectura modular para el desarrollo sencillo de clientes de correo electrónico adaptados a personas ciegas o con problemas severos de visión. Esta arquitectura se basa en la definición de una serie de componentes que luego se pueden añadir, reordenar e intercambiar en el desarrollo de la interfaz de usuario de la aplicación. Para validar y probar la usabilidad de esta arquitectura se ha desarrollado un cliente de correo electrónico adaptado, que presenta la mayoría de funcionalidades existentes en los clientes de correo convencionales.

PALABRAS CLAVE

Arquitectura software, interfaz para invidentes, correo electrónico

1 Introducción

Según la Organización Mundial de la Salud hay aproximadamente 1300 millones de personas con algún tipo de impedimento visual. De estas, 36 millones de personas son ciegas [1].

El uso de tecnología adaptada para estos usuarios es fundamental como parte de su desarrollo personal y su integración en sus entornos sociales y laborales.

Unas de las aplicaciones más utilizadas actualmente son los clientes de correo electrónico. Sin embargo, se trata de elementos complejos que no están bien adaptados a personas ciegas o con impedimentos visuales severos. Los usuarios han de navegar entre múltiples opciones y encontrar los elementos que desean utilizar en cada momento.

Para solucionar esto se han creado algunos sistemas de correo adaptados a personas ciegas [2], [3], [4] y [5]. Sin embargo, estas aproximaciones se basan en aplicaciones específicas que son difícilmente configurables y extensibles.

También es común el uso de aplicaciones basadas en la web [6]. Sin embargo, estos sistemas tienen problemas con los usuarios, perdiendo el foco de la aplicación o limitando su funcionalidad a costa de la mejora de la usabilidad [7].

INTERACCIÓN'19, Junio, 2019, Donostia, España

En esta propuesta presentamos el desarrollo de una arquitectura genérica que permita la creación no sólo de clientes de correo, sino también de otros programas adaptados a las consideraciones y necesidades concretas de los desarrolladores y usuarios. De esta forma, en contraposición a la solución generalizada que representan los lectores de pantalla, permitimos contextualizar la información para que la navegación por la interfaz sea más amigable que con un lector de pantallas.

Además, a partir de esta arquitectura, se ha desarrollado un cliente de correo para validarla y demostrar su funcionalidad. Este cliente de correo intenta ofrecer una amplia funcionalidad atendiendo a las necesidades específicas de sus usuarios. Por ejemplo, es importante establecer una adecuada gestión de contactos y elementos adjuntos o atender cuestiones complejas como trasladar correos entre diferentes carpetas [8].

2 Arquitectura

En lugar de desarrollar un cliente de correo electrónico específico se ha decidido crear una biblioteca genérica que permita la creación sencilla de interfaces adaptadas a las características de cada usuario y aplicación que se quiera desarrollar.

Los componentes de la biblioteca son similares a los componentes de una interfaz gráfica de usuario. Esto es, se basa en elementos como botones, checkboxes, textboxes y listas; con la diferencia principal de que cada uno de estos se presenta de forma adaptada a personas ciegas o con problemas severos de visión.

En la biblioteca todos los componentes actúan de manera genérica, esto es, todos son descritos de forma equivalente mediante el sintetizador de voz.

Como el objetivo de la arquitectura es la creación de una interfaz no visual es necesario extender el paradigma de interfaz convencional a uno adaptado a las nuevas necesidades que se generan. Una persona sin problemas de visión puede obtener de inmediato información de cualquier parte de la interfaz. Sin embargo, un invidente obtiene la información por medio del sintetizador de voz de forma secuencial. Por lo tanto, la navegación por el cliente de correo también se realiza de forma secuencial.

Los elementos destacados en esta arquitectura (que se ha denominado Blue) son: (1) BlueTTS que representa un sistema de texto a voz genérico. (2) BlueComponent que representa un componente como un botón, una lista de elementos seleccionable, etc. (3) BlueComponentList: Representa una lista de componentes. (4) BlueMenu: Representa un menú o, lo que es lo mismo, un contenedor de componentes agrupados bajo un nombre común. Este es el equivalente a las ventanas en una GUI. (5) BlueLabel: Etiqueta que permite trasladar información genérica al usuario. (6) BlueLabelList: Representa una lista de etiquetas. (7) BlueButton: Representa un "botón", una opción que el usuario puede usar para realizar una acción. (8) BlueButtonList: Representa una lista de botones. (9) BlueTextBox: Representa un componente donde el usuario puede introducir texto. (10) BlueCheckBox: Representa un componente que el usuario puede activar o desactivar. (11) BlueTextReader: Permite la lectura de un texto extenso sin bloquear la aplicación, aparte de permitir navegar por las frases que lo conforman.

Esta arquitectura está planteada de la forma en la que los menús (representados en BlueMenu) son componentes contenedores que permiten agrupar de forma secuencial otros componentes, que pueden ser listas (y tener por tanto otros componentes anidados).

Para conseguir mantener la coherencia y mantener un orden de navegación se ha establecido el concepto de foco. Este es análogo al de las interfaces gráficas. El elemento que tiene el foco es el que tiene la atención del usuario en ese momento.

Los lectores de pantalla hacen un uso intensivo del foco para saber qué hay que leer en cada momento. Los elementos están agrupados de forma secuencial en los menús de modo que resulta sencillo cambiar el foco al siguiente elemento.

Como forma de proporcionar una navegación rápida entre componentes el sintetizador debe empezar a describir inmediatamente el elemento que acabe de tomar el foco.

El usuario ha de tener la opción de poder cambiar el foco en cualquier momento. Para esto el sintetizador de voz deba soportar que se le interrumpa en cualquier momento.

Cada uno de los elementos posee una descripción genérica para que el sintetizador de voz informe al usuario. Sin embargo, en el caso de las etiquetas, también se ha definido un mecanismo que permite conseguir descripciones no estáticas. Esto es, cuando una etiqueta obtiene el foco, se ejecuta una rutina que devuelve una descripción que puede cambiar según el estado de la aplicación. Esto sirve, por ejemplo, para obtener información de otros menús de forma eficiente.

El componente BlueTextBox funciona de una manera ligeramente distinta a lo que estamos acostumbrados. En una interfaz gráfica, cuando el textbox recibe el foco, inmediatamente se puede escribir en él. Pero en una interfaz para invidentes, donde la navegación rápida es esencial, es común tener atajos, como por ejemplo presionar una tecla concreta para navegar a un menú concreto sin tener que llegar a su componente botón asociado. Para evitar una interferencia, por ejemplo, que el usuario desee escribir algo y se mueva a un menú diferente, se ha implementado un modo de entrada de información tipo Vim (edición modal).

Esto implica que el BlueTextBox tiene dos modos de funcionamiento, el modo descriptivo y el modo de inserción. En el modo descriptivo se indica la función del textbox y su contenido, y en modo inserción se van leyendo los caracteres que se van insertando o borrando, aparte de recordar al usuario lo que está escrito cuando cambia al modo de inserción. Al cambiar de modo, se indica al usuario en qué modo se encuentra mediante el TTS.

Este tipo de edición modal nos permite tener una gran flexibilidad no sólo con el BlueTextBox (que soporta copiado y pegado de información), sino que permite usar cualquier tecla para realizar otras acciones mientras estemos en modo descriptivo. Esto facilita considerablemente la navegación y ayuda a mitigar uno de los problemas que tienen en ocasiones los programas que funcionan bajo lectores de pantalla: la pérdida del foco por parte de los usuarios.

A raíz del problema de la perdida de foco se ha implementado un mecanismo que permite tener atajos para reencontrar el foco. Por ejemplo, se puede permitir que pulsando una tecla por defecto se realice una descripción del menú y el elemento en el que está el usuario. O también que se pueda acceder a una ayuda contextual pulsando otra tecla. Esta ayuda contextual se ha diseñado de modo que esté ligada a cada menú.

También se ha diseñado un sistema de perfiles que permite cambiar la posición de los elementos de un menú y mantenerla guardada para un posterior uso. Por ejemplo, imaginemos un escenario en el que el usuario está acostumbrado a un programa que tiene los elementos en una determinada posición. Con esta arquitectura es posible cambiar los elementos de un menú de posición de forma que el foco resida primero en el elemento que más le interese al usuario. Esta configuración se puede guardar fácilmente y restablecerse de forma transparente al usuario.

Todos esos componentes son fácilmente agregables a la aplicación que se desee crear. El sistema se asegura de que cada elemento tenga un identificador único, a excepción de elementos potencialmente dinámicos como los botones de una BlueButtonList. Esto es útil a la hora de cambiar los elementos de posición, y también permite excluir a los elementos dinámicos del sistema de cambio de posición.

En definitiva, esta arquitectura permite acceder a los elementos de la aplicación de forma secuencial, en contraposición a las interfaces gráficas de usuario. Por medio de técnicas como la interrupción al sistema de texto a voz y atajos de teclado, el usuario puede recorrer rápidamente la interfaz.

3 Aplicación de correo

Para la creación de este cliente de correo se han tenido en cuenta una serie de consideraciones previas que se han establecido como base para el desarrollo de la aplicación.

En primer lugar, se ha decidido realizar un cliente que trabaje como una capa adicional sobre el sistema de correo de Gmail (aunque teóricamente es capaz de operar con cualquier servidor que soporte IMAP y SMTP seguros).

En segundo lugar, se ha establecido que la aplicación de cliente de correo no requiera de ningún tipo de instalación y pueda ejecutarse en cualquier ordenador personal, independientemente de su sistema operativo. Esto permite, por ejemplo, llevar siempre el cliente de correo instalado en un pendrive y utilizarlo de forma sencilla y segura allá donde sea necesario.

El cliente de correo se basa en el concepto de secuencialidad a la hora de navegar y mostrar la información tal y como ya se ha explicado. Los diferentes componentes se muestran de forma secuencial, según una secuencia establecida.

Para avanzar por cada una de los componentes de la secuencia se ha de utilizar la tecla Tab. También es posible retroceder al componente anterior utilizando la combinación de teclas Shit+Tab.

Además, se han establecido una serie de teclas predefinidas a modo de funcionalidades especiales y atajos. Estas son: (1) La tecla R para que el sintetizador de voz repita la información de la etiqueta correspondiente al componente que tiene actualmente el foco. (2) La tecla W ofrece información contextual del menú y el elemento en el que se encuentra el foco. (3) La tecla H ofrece ayuda contextual adicional asociada a cada menú. (4) La tecla Control permite iniciar y parar la inserción de texto cuando el usuario se encuentre en un componente de tipo inserción de texto o textbox. (5) La tecla Enter permite "presionar un botón", esto es, realizar una acción. (6) Las teclas cursor permiten subir y bajar por los elementos de tipo lista, tales como carpetas de correo, mensajes, etc. (7) La tecla N permite acceder directamente a la secuencia de envío de un nuevo correo electrónico.

La primera vez que se ejecuta el cliente se inicia un pequeño tutorial que explica cómo navegar por la interfaz. A continuación, se entra en un menú de encriptación que permitirá encriptar toda la información de la aplicación con una contraseña maestra. Al tratarse de un cliente portátil y sin instalación se ha considerado necesario que toda la información se almacene de forma suficientemente segura. En concreto se ha usado un algoritmo de cifrado denominado Blowfish, debido a su alta disponibilidad en cualquier sistema operativo y su eficiencia en tiempo de ejecución [9].

A partir de ahí, o en sucesivas ejecuciones, se accede directamente al menú de acceso. En este menú se puede configurar, si no se ha hecho antes, los datos de la cuenta de correo: servidores de entrada, salida, usuarios y contraseñas.

Ya dentro del cliente de correo se entra en el denominado Menú de carpetas. En este menú se empieza en la carpeta Inbox informando del número de correos sin leer. El usuario puede desplazarse, utilizando los cursores, por las diferentes carpetas de correo o acceder a los correos de la carpeta donde se encuentre posicionado. Desde aquí se puede acceder al menú de contactos, el menú de lectura de mail o el de nuevo correo.

En el menú de contactos se puede navegar por la lista de contactos, así como añadir un nuevo contacto.

El menú de lectura de mail se irá desplazando por cada uno de los elementos que componen un correo electrónico. Estos son el asunto del mensaje, remitente, destinatarios, cuerpo del mensaje y ficheros adjuntos. Además, si los mensajes están agrupados en formato conversación se puede avanzar o retroceder a los mensajes posteriores o anteriores de la conversación. Por último, estarán las opciones de responder o reenviar el mensaje.

El menú de escritura de email se desplazará por las opciones destinatarios, con copia, con copia oculta, asunto, cuerpo del mensaje, añadir adjuntos y enviar correo.

Siguiendo un ejemplo sencillo de un usuario situado en la carpeta inbox que lee el primer correo y responde, los pasos que se deberían seguir son los siguientes:

El usuario pulsa la tecla Tab y se posiciona en la lista de emails. Por defecto, al tomar el foco una lista, toma el foco el primer elemento anidado en la lista. Se lee al usuario el asunto del correo. El usuario pulsa Enter y se le avisa de que el correo se está abriendo. Una vez abierto, se avisa al usuario de que se encuentra en el menú de lectura de correo. Por defecto, al tomar el foco un menú lo hace el primer componente de ese menú, en este caso una etiqueta (BlueLabel) que contiene el asunto del correo. Esta se lee al usuario. El usuario pulsa Tab y toma el foco una etiqueta que contiene los destinatarios y las direcciones en copia. Se leen al usuario, indicando que son los receptores. El usuario pulsa Tab y toma el foco otra etiqueta que contiene el emisor, que de nuevo se lee al usuario, indicando que es el emisor. Una vez acabada la lectura del emisor, el usuario pulsa Tab de nuevo y comienza la lectura del cuerpo, contenido en un BlueTextReader. Por defecto, al tomar el foco este componente, se empieza a leer el texto. El usuario puede, en cualquier momento, pausar la lectura con la tecla S, repetir la última frase con R y moverse por las frases mediante los cursores, como si de una lista se tratara. Si no realiza estas acciones la lectura continúa de forma transparente al usuario. El efecto que se consigue con este componente es el de parecer una etiqueta hasta que se le interrumpe, momento en el que se muestra su capacidad de navegación secuencial, siendo un caso particular de una BlueLabelList, en la que cada frase es una BlueLabel. El componente además

consigue ser eficiente en tiempo de ejecución mediante técnicas de threading, esto es, es capaz de detectar qué frase está leyendo en cada momento y enviar al sintetizador sólo el fragmento necesario en cada circunstancia.

Una vez terminada la lectura del cuerpo del email, el usuario pulsa Tab para posicionarse en la lista de adjuntos. El usuario navega con los cursores sobre los ficheros adjuntos y pulsa Enter cuando se encuentra en el que desee descargar. Si descarga un fichero, se le indica que se va a descargar y, una vez finalizado, se le avisa mediante una callback.

Al pulsar Tab, en caso de que no haya emails siguientes ni anteriores en la conversación, tomaría el foco el botón de responder. Si hubiera emails siguientes o anteriores tomaría el foco el botón de siguiente en conversación, y al pulsarlo recargaría el menú de lectura con los datos del email siguiente, procediendo nuevamente a tomar el foco el principio del menú (se leería el asunto).

Si se decide responder o reenviar, el usuario entra en el menú de escritura de correo, con los datos oportunos rellenos (en el caso de la respuesta, el asunto y el destinatario queda relleno, y en el caso del reenvío, el asunto, cuerpo y adjuntos quedan rellenos).

Este es un ejemplo sintético de un caso de uso determinado. El cliente de correo desarrollado soporta la mayor parte de las funcionalidades presentes en un cliente de correo convencional, como pueden ser desplazamiento entre carpetas, mensajes y conversaciones, procesar archivos adjuntos, mover mensajes entre carpetas, etc.

El proceso de navegación se ha realizado de forma que resulte intuitiva y sencilla. El acceso a elementos del mismo tipo se hace de forma análoga. Y además se proporciona ayuda contextual y atajos para que usuarios con menos experiencia o más avanzados puedan navegar por la interfaz según sus necesidades.

4 Conclusiones y trabajo futuro

En este artículo hemos presentado una arquitectura para la creación de clientes de correo electrónico adaptadas a personas ciegas o con impedimentos severos de visión.

La arquitectura se ha desarrollado de modo que la creación de estos clientes de correo sea lo más sencilla posible. Además, se ha definido con el objetivo de que los clientes desarrollados sean modulares, configurables y adaptados a las necesidades concretas que se establezcan.

A partir de esta arquitectura se ha creado un cliente de correo para demostrar su viabilidad y utilidad. El cliente de correo está adaptado a las necesidades de navegación de las personas ciegas. Pero a su vez intenta maximizar su funcionalidad, para ser lo más cercana posible a la de un cliente de correo convencional.

Aunque la parte correspondiente a este trabajo ha sido la creación de una arquitectura sencilla y modular, el objetivo final es la creación de clientes de correo adaptados a los usuarios finales. Es por esto que como trabajo futuro quedaría validar la propuesta de cliente de correo desarrollada con usuarios reales.

También hay otros aspectos que mejorar en esta propuesta. Concretamente, el soporte para ciertos caracteres especiales en diferentes sistemas operativos, mejoras en el parsing de los hipervínculos para pasarlos al sintetizador de voz, mejoras en el componente de lectura de textos largos, pruebas de integración de diferentes sintetizadores de voz, capacidad de cambiar el idioma en tiempo de ejecución, integración con la configuración del lector de pantalla del usuario (permitiendo un descubrimiento en tiempo real del lector de pantalla) o añadir la posibilidad de describir imágenes mediante técnicas de reconocimiento de imágenes.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el Proyecto “e-Madrid: Investigación y desarrollo de tecnologías educativas en la Comunidad de Madrid” (P2018/TCS-4307).

REFERENCIAS

[1] Bourne RRA, Flaxman SR, Braithwaite T, Cicinelli MV, Das A, Jonas JB, et al.; Vision Loss Expert Group. Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis. Lancet Glob Health. 2017 Sep;5(9):e888–97.

[2] Akif Khan1, Shah Khusro, Badam Niazi, Jamil Ahmad, Iftikhar Alam, Inayat Khan. 2018. TetraMail: a usable email client for blind people. Universal Access in the Information Society, September 2018. https://doi.org/10.1007/s10209-018-0633-5

[3] K.V.N. Sunitha, N. Kalyani. 2010. VMAIL Voice Enabled Mail Reader. International Conference on Recent Trends in Information, Telecommunication and Computing.

[4] Hari Priya S L, Karthigasree S, Revathi K. 2015. Voice –Based E-Mail (V-Mail) for blind. International Journal of Scientific Research in Science, Engineering and Technology.

[5] Apoorva Kale, Ashwini Jenekar, Shraddha Kapse, Romi Taskar, Shruti Menon. 2018. Voice Based Email System. International Journal on Future Revolution in Computer Science & Communication Engineering. 4, 3.

[6] G.Shoba, G.Anusha, V.Jeevitha, R.Shanmathi. 2014. An interactive email for visually impaired. International Journal of Advanced Research in Computer and Communication Engineering, 3, 1.

[7] Brian Wentz, Jonathan Lazar. 2011. Usability Evaluation of Email Applications by Blind Users. Journal of Usability Studies, 6, 2, 75-89.

[8] Brian Wentz, Harry Hochheiser, J. Lazar. 2010. Email Usability for Blind Users. Designing Inclusive Interactions: Inclusive Interactions Between People and Products in Their Contexts of Use, 197-206.

[9] Priyadarshini Patil, Prashant Narayankar, Narayan D.G, Meena S.M. 2016. A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. Procedia Computer Science, 78, 617-624.